

BPAAS MONITORING AND EVALUATION PROTOTYPES

D3.6

Editor Name	Kyriakos Kritikos (FORTH)
Submission Date	June 30, 2017
Version	1.0
State	Final
Confidentially Level	PU



Co-funded by the Horizon 2020
Framework Programme of the European Union

EXECUTIVE SUMMARY

BPaaS Evaluation is a major BPaaS lifecycle phase focusing on deriving added-value knowledge, by performing different types of analysis, which assist the broker in further improving the BPaaS offered. As such, the previous version of this deliverable, D3.5 [1], reported on BPaaS evaluation modelling and analysis blueprints in form of research ideas or early prototypes. Following the work conducted in the final 6-month period of T3.3, some of the research ideas were transformed into proof-of-concept prototypes while the existing prototypes improved their technological readiness level. In both cases, most respective research results were integrated into the CloudSocket prototype implementation in WP4 which outlines the importance, maturity level and potential impact of the work conducted. In fact, all research prototypes are complete enough to be extended to build new research prototypes or to enhance existing products by supplying the missing BPaaS analysis capabilities.

This deliverable aims at supplying an update over all blueprints analysed in D3.5 by also explicating which were finally integrated into the CloudSocket prototype. Apart from the BPaaS Evaluation prototype analysis, this deliverable reports on the update of the *Monitoring Engine* developed for T3.2, which was enhanced with the ability to support different kinds of Time Series Data Bases (TSDBs) that exhibit advanced storage and analysis capabilities. The main rationale for including this update is that as measurement information is critical for the different types of analysis supported in the BPaaS Evaluation research prototype, it is important to also keep track of the modifications done on the respective engine supplying such measurement information.

By considering the BPaaS evaluation modelling prototypes, some extensions have been performed. The main structure of the BPaaS evaluation ontology, i.e., the semantic meta-model capturing the BPaaS dependency hierarchy, was slightly modified to cover some details, mainly for infrastructural services. On the other hand, the OWL-Q [2] KPI extension [3], i.e., the semantic KPI meta-model, was extended with additional features that were outlined in the future work directions in D3.5. Moreover, domain code supporting OWL-Q was also generated focusing on the parsing and serialisation of OWL-Q models, enabling the appropriate management of OWL-Q models and their exploitation for various kinds of semantic reasoning tasks, like non-functional service matching.

The overall architecture of the BPaaS Evaluation research prototype was stable; however, internal architecture modifications for most of its main components were applied in sync with updates on their implementation. Moreover, this prototype became multi-tenant, able to support multiple brokers at the same time. The component naming was also changed from modules to engines to signify that a kind of processing and analysis is performed in each main component. Concerning each component individually, the *Harvesting Engine* (previously named as *Harvester*) was updated to completely produce the right, semantically-lifted and linked information for the different kinds of analysis offered by the research prototype. This is also the main component contributing to the multi-tenant capability as it is able to store broker-based information in partitioned, broker-specific RDF graphs. The remaining components just take the broker name as input to know on which partition to perform the respective analysis needed. The *Conceptual Analytics Engine* was updated to support different kinds of KPI drill-down while KPI evaluation now relies either on fixed KPIs or dynamically specified KPI metric formulas and contexts. A capability to store KPI evaluation results was also realised to enable the immediate and fast querying of the BPaaS evaluation history. The *Deployment Discovery* and *Pattern Discovery Engines* were transformed into first research prototypes able to derive best BPaaS deployments and event patterns leading to KPI violations, respectively. Finally, the *Process Mining Engine* was completed to support the execution of state-of-the-art process mining algorithms focusing on process model re-creation based on the BPaaS execution history.

We do plan to evolve the BPaaS Evaluation research prototype in the near future based on the directions outlined in D3.5. We do see its added-value especially from the side of the broker and we really believe in its potential. Thus, our future effort will concentrate on enhancing it to make it an advanced evaluation companion to the broker towards the pursuit of improving the BPaaS offered and increasing the gains drawn out of them.

PROJECT CONTEXT

Workpackage	WP3: Business Process as a Service Research
Task	T3.3: BPaaS Evaluation Environment Research
Dependencies	Input to WP4

Contributors and Reviewers

Contributors	Reviewers
Kyriakos Kritikos, Chrysostomos Zeginis, Daniel Seybold, Frank Griesinger	Robert Woitsch, Yongzheng Liang, Frank Griesinger

Approved by: Stefan Wesner (UULM) as WP 3 Leader

Version History




Version	Date	Authors	Sections Affected
0.1	May 04, 2017	Kyriakos Kritikos (FORTH)	Initial version, TOC
0.1_FORTH	May 11, 2017	Kyriakos Kritikos (FORTH)	Initial content - FORTH Contribution
0.1_ULM	May 17, 2017	Daniel Seybold (UULM), Frank Griesinger (UULM)	Initial content - UULM Contribution
0.2	May 18, 2017	Kyriakos Kritikos (FORTH)	1st complete draft
0.2_ULM	May 22, 2017	Daniel Seybold (UULM), Frank Griesinger (UULM)	Update of ULM contribution
0.3	May 26, 2017	Kyriakos Kritikos (FORTH)	Complete draft ready for review
0.31	June 2, 2017	Robert Woitch (BOC), Yongzheng Liang (BWCON), Frank Griesinger	Reviewed document
1.0	June 20, 2017	Kyriakos Kritikos	Pre-final version
1.0-a	June 23, 2017	Kyriakos Kritikos (FORTH), Daniel Seybold (UULM)	Final version

Copyright Statement – Restricted Content

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

This is a restricted deliverable that is provided to the community under the license Attribution-No Derivative Works 3.0 Unported defined by creative commons <http://creativecommons.org>

You are free:

	to share within the restricted community — to copy, distribute and transmit the work within the restricted community
Under the following conditions:	
	Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
	No Derivative Works — You may not alter, transform, or build upon this work.

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work.

This is a human-readable summary of the Legal Code available online at:

<http://creativecommons.org/licenses/by-nd/3.0/>

TABLE OF CONTENT

1	Introduction and Problem Statement	9
1.1	Introduction	9
1.2	Project Context & Blueprint Selection	9
1.3	Running Example.....	10
1.4	Structure	11
2	BPaaS Monitoring prototype.....	12
2.1	Features.....	12
2.2	Architecture.....	13
2.3	Set-up	14
2.4	Future Work	15
3	BPaaS Modelling Prototypes	16
3.1	Evaluation Ontology.....	16
3.1.1	Running Example Application.....	17
3.1.2	Features	18
3.1.3	Implementation	19
3.1.4	Set-up.....	19
3.1.5	Future Work.....	19
3.2	OWL-Q KPI Extension	19
3.2.1	Running Example Application.....	20
3.2.2	Features	21
3.2.3	Implementation	23
3.2.4	Set-up	23
3.2.5	Future Work.....	23
4	Evaluation Environment Analysis Prototypes	24
4.1	Harvesting Engine.....	25
4.1.1	Running Example Application.....	26
4.1.2	Features	26
4.1.3	Architecture.....	27
4.1.4	Implementation & Set-up	28
4.1.5	Future Work.....	28
4.2	Conceptual Analytics Engine	28
4.2.1	Running Example Application.....	29
4.2.2	Features	30
4.2.3	Architecture.....	30

4.2.4	Set-up	31
4.2.5	Future Work.....	32
4.3	Deployment Discovery Engine	32
4.3.1	Running Example Application	33
4.3.2	Features	35
4.3.3	Architecture.....	35
4.3.4	Set-up	36
4.3.5	Future Work.....	37
4.4	Pattern Detection Engine	37
4.4.1	Running Example Application	37
4.4.2	Features	39
4.4.3	Architecture.....	39
4.4.4	Set-up	40
4.4.5	Future Work.....	40
4.5	Process Mining Engine	40
4.5.1	Running Example Application	41
4.5.2	Features	41
4.5.3	Architecture.....	42
4.5.4	Set-up	43
4.5.5	Future Work.....	43
5	Conclusion	44
5.1	Conclusion	44
5.2	Future Work	45
5.3	Final BPaaS Research Conclusion	45
6	References	48
Annex A:	List of Abbreviations	49

LIST OF FIGURES

Figure 1: The BPMN model for the Send Invoice BPaaS	10
Figure 2: The detailed BPMN workflow model for the Send Invoice BPaaS.....	11
Figure 3: The architecture of the InfluxDB Monitoring Engine prototype	14
Figure 4: The extension of the evaluation ontology focusing on introducing additional infrastructural aspects	16
Figure 5: A snapshot of the Evaluation Ontology model for the Send Invoice BPaaS.....	17
Figure 6: A snapshot of the OWL-Q model focusing on the definition of the MeanCycleTimeKPI	21
Figure 7: The update of the OWL-Q ontology in particular with respect to the KPI extension	22
Figure 8: The overall architecture of the BPaaS Evaluation research prototype	25
Figure 9: Example of RDF triple linkage by Harvester.....	26
Figure 10: The architecture of the Harvesting Engine	27
Figure 11: Snippet of the SPARQL query used for the evaluation of the Mean Cycle Time KPI.....	30
Figure 12: The architecture of the Conceptual Analytics Engine	31
Figure 13: A snapshot of the results of the execution of Deployment Discovery Engine on the running example	35
Figure 14: The architecture of the Deployment Discovery Engine.....	36
Figure 15: The architecture of the Pattern Detection Engine.....	40
Figure 16: The workflow model mined from the execution log.....	41
Figure 17: The architecture of the Process Mining Engine	42

LIST OF TABLES

Table 1: Monitoring Engines feature comparison	13
Table 2: Performance variations for the BPaaS of the running example	34
Table 3: Simple adaptation rules defined for the BPaaS of the running example.....	38
Table 4: Small trace log for the events associated with the BPaaS of the running example	38

1 INTRODUCTION AND PROBLEM STATEMENT

1.1 Introduction

BPaaS Evaluation is a major phase in the BPaaS lifecycle as it deals with the derivation of added-value knowledge which can assist in the improvement of a certain BPaaS. Such a phase can be supported with different types of analysis which span the evaluation of KPIs and their drill-down, the discovery of best deployments for BPaaSes, the detection of event patterns that lead to the violation of KPIs and the execution of process mining algorithms to find potential discrepancies and performance bottlenecks within a certain BPaaS. All such types of analysis require the use of a system which delivers high-quality and suitably semantically linked information in order to raise the accuracy of the analysis tasks performed on it.

1.2 Project Context & Blueprint Selection

In the context of the CloudSocket project and the WP3 workpackage, Task 3.3 deals with conducting research and development work which should result in the production of corresponding research prototypes that support the realisation of BPaaS evaluation analysis tasks. Indeed, the first deliverable, D3.5, of this task was dedicated to describing a set of BPaaS evaluation modelling and analysis blueprints which were in the form of research ideas or early prototypes. Such blueprints attempted to cover the aforementioned analysis tasks as well as the need to appropriately harvest and semantically link the corresponding information on which the analysis will rely. They have also included modelling blueprints in the form of ontology prototypes which focused on prescribing the right structure to which the evaluation information gathered and semantically uplifted should conform.

Based on the conducted work in the last 6 months of the project within T3.3, most of these blueprints have been transformed into research prototypes which were finally selected to be integrated into the final implementation of the BPaaS Evaluation Environment. These prototypes were selected according to their importance in supporting the various types of analysis intended to be delivered by the BPaaS Evaluation Environment in the CloudSocket implementation [4]. However, the level of maturity as well as the required level of integration played a major role in this selection. To this end, while all of the prototypes did have the right maturity level, the *Pattern Detection Engine* was decided to be excluded from the integration in the final BPaaS Evaluation Environment prototype. The *Pattern Detection Engine* has the main goal to detect event patterns that could lead to KPI/SLO violations. However, by considering the ability to perform KPI drill-down and the current adaptation capabilities of the CloudSocket prototype, this integration loss is not so critical. After all, many types of analysis are do offered which will certainly make the life of the broker much easier.

In this respect, in the end, all BPaaS evaluation blueprints were implemented and are available for downloading and for any possible extension. This includes the following blueprints, now offered in the form of research prototypes, which are summarised as follows:

- The *Evaluation Ontology* [1] modelling prototype which is a semantic meta-model that captures the BPaaS dependency hierarchy along with additional information spanning the organisational and adaptation aspects;
- The *OWL-Q KPI Extension* [3] modelling prototype which is a semantic meta-model focusing on the modelling of KPIs and their dependencies, the association of KPIs to organisational goals, the modelling of both manual and automatic measurements along with their assessment information as well as the enrichment of metric formula input with the capability to draw information from external information sources.

- The *Harvesting Engine* evaluation prototype which enables to harvest information from multiple information sources within the CloudSocket prototype, to semantically structure it and link it as well as to store it in a *Semantic Knowledge Base (Semantic KB)*.
- The *Conceptual Analytics* analysis prototype which offers two different forms of KPI analysis: (a) KPI evaluation; (b) KPI drill-down as well as some utility functions.
- The *Deployment Discovery* analysis prototype which enables to discover the best deployment(s) for a BPaaS based on its execution history or the execution history of other BPaaSes that are similar or equivalent to it.
- The *Pattern Detection* analysis prototype which enables to detect event patterns that lead to SLO/KPI violations and to also map them to adaptation rules that could be used to enhance the adaptive behaviour of a BPaaS.
- The *Process Mining* analysis prototype which enables to execute state-of-the-art process mining algorithms in order to find BPaaS discrepancies with respect to the designed and the effective BPaaS workflow model.

During this 6 month period, work on T3.2 continued. As this work is considered a prerequisite for successful operation for many types of analysis supported by the evaluation prototypes, it was decided to include an analysis of its update in this deliverable in order to unveil certain aspects that might also lead to potential future improvements in some evaluation prototypes (e.g., the Harvesting Engine prototype). That work mainly concerned the update of the *Monitoring Engine* to support multiple Time Series DataBases (TSDBs) instead of only KairosDB¹. This enables exploiting some advanced storage & analysis features included in these new TSDBs. Please note that this *Monitoring Engine* has already been integrated into the BPaaS Execution Environment implementation.

1.3 Running Example

The Send Invoice use case ([5], [6]) was exploited as a real BPaaS in order to showcase the main benefits for the evaluation prototypes analysed. This use case concerns the implementation of a BPaaS which connects to the CRM of the BPaaS customer, obtains client information and then exploits this information in order to build invoices and send them via email. This BPaaS will need to exploit two different kinds of functionalities: (a) CRM management and (b) invoice management. Such functionalities can be drawn either from the cloud (i.e., in the form of external SaaS) or could map to already developed or purchased software of the broker (i.e., in the form of mainly internal SaaS). The process model at the business level for this BPaaS is shown in Figure 1.

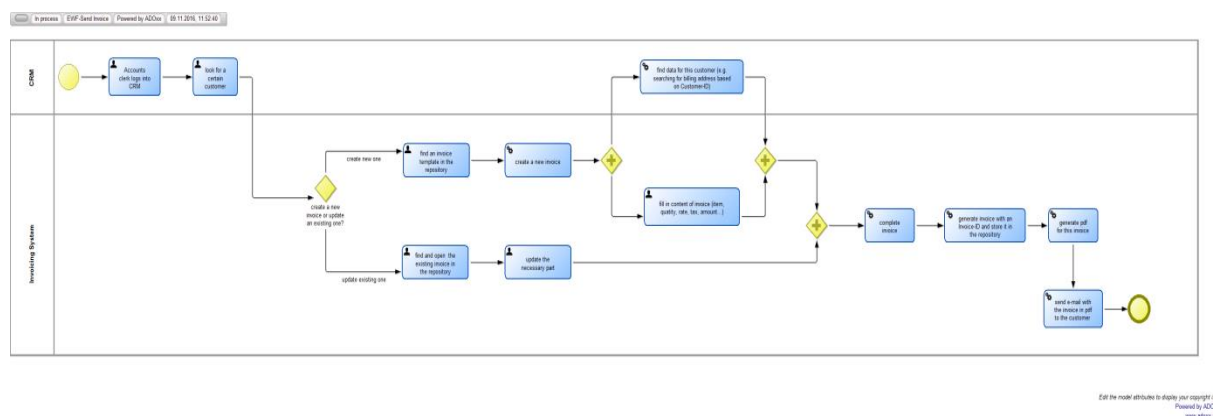


Figure 1: The BPMN model for the Send Invoice BPaaS

The detailed workflow model in BPMN for this BPaaS is also depicted in Figure 2 below.

¹ <https://kairosdb.github.io/>

2 BPAAS MONITORING PROTOTYPE

The exploited Monitoring Engine in the Execution Environment [8], comprising the KairosDB time series database² in conjunction with Cloudiator components Visor³ and AXE⁴, focuses on the storage and processing of typical non-functional metrics at the IaaS level, e.g., CPU utilisation or memory usage or disk usage, at the SaaS level, e.g., served requests or executed queries per time period, as well as at the BPaaS level, e.g., successful sent invoices. As the Monitoring Engine bridges the gap between the Execution Environment and the Evaluation Environment by providing raw and aggregated monitoring data of the deployed BPaaS Bundles which can be exploited for KPI evaluation purposes, the evolution of the Monitoring Engine affects the Evaluation Environment and its capabilities. To this end, we now analyse the developments performed in the Monitoring Engine in the final 6-month period of WP3.

In the context of the outlined research directions of the Evaluation Environment Prototypes [7], an internal analysis of the current Monitoring Engine with respect to the prototype requirements was performed as the landscape of available time series databases (TSDBs) has evolved recently over the last years. While former TSDBs provide basic statistical calculations on the stored data, such as average, min or max, more recent TSDBs provide enhanced features, such as continuous calculation, i.e., continuously computing (aggregation) functions based on the input data and storing the results on long-term storage, thus providing scheduled mechanisms to reduce historical data by aggregation functions.

In order to better support the Evaluation Environment prototypes, the Monitoring Engine is extended to support more powerful TSDBs, following the *Generic TSDB Layer* concept presented in D3.3 [9]. Therefore, the Monitoring Engine is extended to support the InfluxDB⁵ TSDB, which gained a lot of momentum in the recent years and provides advanced features, such as continuous queries for continuous calculations, mechanisms to support long-term storage and a powerful set of algorithmic operations, which can be exploited by the BPaaS Evaluation Environment. A detailed overview of the enhanced features of the Monitoring Engine prototype based on InfluxDB in comparison to the current Monitoring Engine based on KairosDB is provided in the following.

2.1 Features

In order to analyse the current Monitoring Engine and its capabilities with respect to possible alternative TSDBs, the recently published survey on available TSDBs [10] provided a solid knowledge base to guide the selection of a more advanced TSDB, namely InfluxDB. The following table provides an overview of the evolved Monitoring Engine prototype (based on InfluxDB) and the current Monitoring Engine (based on KairosDB).

Feature	Monitoring Engine (based on KairosDB)	Monitoring Engine Prototype (based on InfluxDB)
---------	---------------------------------------	---

² <https://kairosdb.github.io/>

³ <https://github.com/cloudiator/visor>

⁴ <https://github.com/cloudiator/axe-aggregator>

⁵ <https://docs.influxdata.com/influxdb/v1.2/introduction/>

Version	1.1.3	1.2.2
Basic operations	insert, read, scan, average, sum, count, max, min	insert, read, scan, average, sum, count, max, min, update, delete
Advanced operations	grouping, filtering	grouping, filtering, transformations, predictors
Continuous calculations	no	continuous queries
Downsampling	yes	yes
High availability	only at the storage layer by using Apache Cassandra, not at the computation layer	yes, by using InfluxDB Relay
Long-term storage	no	yes, by using continuous queries and retention policies
Visualisation	basic web UI, no real-time view	advanced web UIs based on Chronograf or Grafana with a real-time view
Usage	REST API, client libraries for Java	REST API, client libraries for Java, Go, python, Haskell, PHP, Javascript,
Plug-ins	no	Kapacitor, Chronograf, Telegraf

Table 1: Monitoring Engines feature comparison

2.2 Architecture

As the current Monitoring Engine is coupled with components of the Cloudiator Framework, the architecture of the enhanced InfluxDB Monitoring Engine prototype requires extensions to the Cloudiator components in order to support the BPaaS monitoring lifecycle. Figure 3 shows a high-level architecture of the Monitoring Engine prototype, including the extended Cloudiator components, namely Colosseum, Visor and the AXE aggregator.

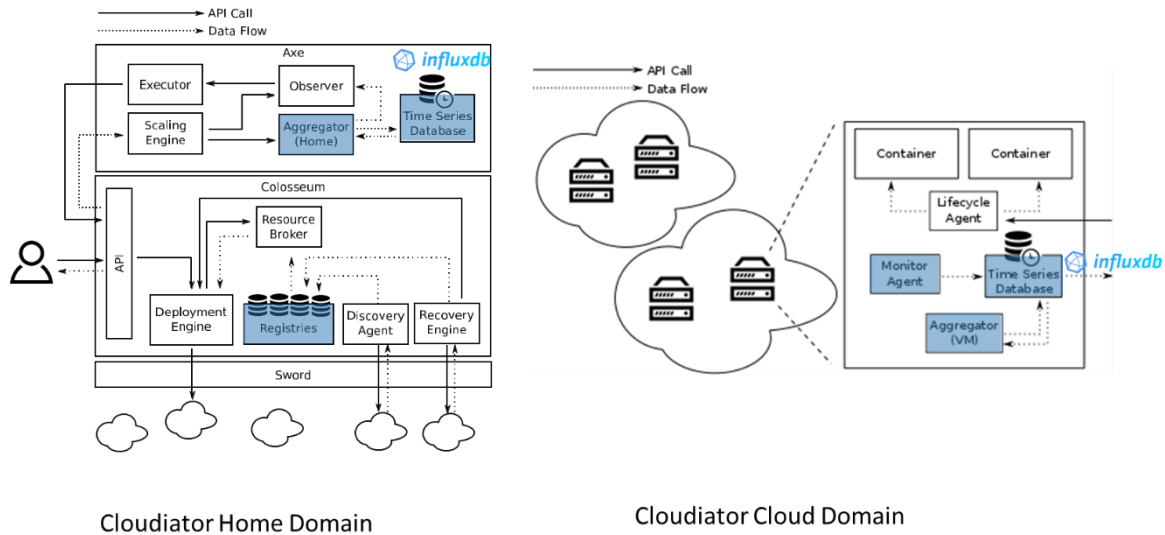


Figure 3: The architecture of the InfluxDB Monitoring Engine prototype

Figure 3 highlights with the blue colour (over components represented by the corresponding boxes) the extension to the existing Cloudiator framework, including the Monitoring Engine as described in D4.6-4.8 [4].

The internal registries of Cloudiator component Colosseum are extended to support the orchestration of multiple InfluxDB instances across the Cloudiator Home Domain and the Cloud Domain, where local InfluxDB instances reside on the provisioned VMs.

The Monitoring Agent, namely Visor, is extended in order to support the reporting to InfluxDB. Therefore, the InfluxDB reporting is customised to support the CloudSocket-specific metadata.

The aggregators, running in the Home and Cloud domain are extended as well, to support the aggregation requirements in CloudSocket that can be satisfied with the more advanced aggregation capabilities of InfluxDB that have been incorporated.

The exploitation of the InfluxDB Monitoring Engine prototype by the Evaluation Environment is eased by InfluxDB, as a REST API and a variety of client libraries are supported, enabling the seamless usage of the advanced features of InfluxDB presented in the previous section.

2.3 Set-up

The setup of the Monitoring Engine prototype requires the extended version of the Cloudiator components Colosseum, AXE aggregator and Visor and an instance of InfluxDB. The sources of the Cloudiator extensions as well as those of InfluxDB are publicly available on GitHub⁶. A detailed setup and usage guide can be found in the CloudSocket Wiki⁷.

The dashboard of the running instance is available at <http://134.60.64.166:8888> and the REST API at <http://134.60.64.166:8086>. The InfluxDB Monitoring Engine prototype will be referenced on the CloudSocket webpage of the Innovation Shop.

⁶ <https://github.com/cloudiator/>

⁷ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Monitoring+Engine+%28InfluxDB+Prototype%29>

2.4 Future Work

The extensible architecture of the Monitoring Engine provides the option to integrate additional TSDBs, such as Prometheus⁸ or Druid⁹, in order to support an even more flexible management of the BPaaS Bundle monitoring data. Such alternatives will be explored and a possible extension will be performed in the near future for the Monitoring Engine in order to become even more capable of incorporating additional realisation capabilities which could meet the diverse requirements of the CloudSocket brokers.

⁸ <https://prometheus.io/>

⁹ <http://druid.io/>

3 BPAAS MODELLING PROTOTYPES

The BPaaS Evaluation Environment has relied on two modelling blueprints that have been transformed into corresponding prototypes: (a) the evaluation ontology which enables the capturing of BPaaS dependency hierarchies; (b) the KPI ontology which enables the modelling of KPIs. Both prototypes are needed for the semantic structuring and linking of dependency and monitoring information that is collected from the Execution Environment by the *Harvesting Engine* in order to support the main analysis functionality exposed by the BPaaS Evaluation Environment.

In the following, we present the two modelling prototypes in two main sub-sections which have the same structure. This structure involves elaborating over the prototype features, explaining the prototype implementation, describing the set-up of the prototype and finally determining some promising directions for future work for this prototype, outside the context of this project.

3.1 Evaluation Ontology

The Evaluation Ontology [1] has been mainly developed in order to capture the dependencies involved in the hierarchy of a certain BPaaS. Such dependencies cover multiple levels and concern different types of entities. Through the modelling of such dependencies, we are able to continuously record the current state of a BPaaS as well as the way it evolves over time, thus also catering for a kind of *models@runtime* [11] approach. Such an approach could be highly suitable for the semantic runtime management of cloud-based BPs, i.e., BPaaSes.

Since its initial conception and realisation, this ontology has been slightly evolved in order mainly to cover additional infrastructural information. This includes some image-related information for IaaS instances, the modelling of clouds as well as their providers, as sub-notions of tenants. Respective relationships between these new notions were introduced while also between them and some original notions. Figure 4 shows the small extension of the ontology that has been developed during this last 6 month period of T3.3.

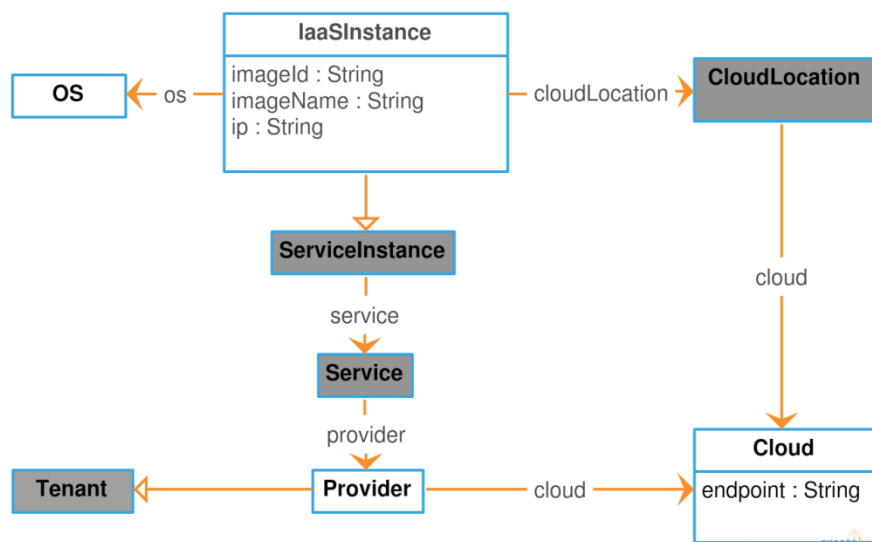


Figure 4: The extension of the evaluation ontology focusing on introducing additional infrastructural aspects

Apart from this extension, the rest of the ontology content has more or less remained untouched. This was also validated while using the ontology in the context of other BPaaS Evaluation components which enabled us to obtain the respective positive feedback that included the requirement to slightly modify the content of this ontology.

3.1.1 Running Example Application

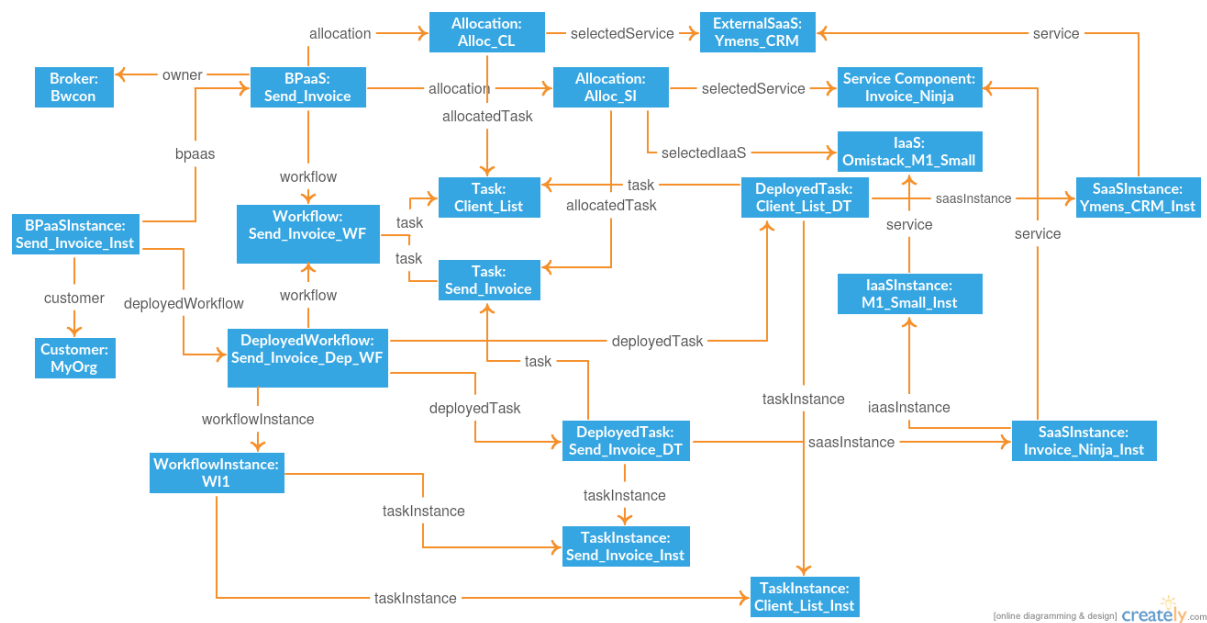


Figure 5: A snapshot of the Evaluation Ontology model for the Send Invoice BPaaS

Figure 5 shows how the running example can be modelled based on the *Evaluation Ontology*. This figure depicts both the type and instance level. At the type level, you can see that the Send Invoice BPaaS is connected to a certain workflow and to a specific broker (BWCON)¹⁰. The workflow is connected, in turn, to a set of tasks, where we have included only the service-based ones for space economy reasons. The BPaaS is also associated with a set of allocations that specify which workflow tasks have been mapped to which cloud services. Thus, as it can be seen, task *Client List* has been mapped to the YMENS CRM¹¹ external SaaS service. On the other hand, task *Send Invoice* has been mapped to an internal service component which has to be deployed on an IaaS service (Omistack m1.small on the UULM infrastructure).

At the instance level, the BPaaS is related with a BPaaS instance (*Send_Invoice_Inst*). The latter is associated with a certain customer named as *MyOrg* and a specific deployed workflow (*Send_Invoice_Dep_WF*). The deployed workflow has one or more deployed tasks. Such tasks are allocated to certain service instances. For example, the deployed task *Client_List_DT* has been allocated to an instance of the YMENS CRM SaaS (*Ymens_CRM_Inst*). The deployed workflow is also associated with one or more workflow instances which represent the actual instances that have been executed by *MyOrg*. Each of these workflow instances, like *WI1*, map in turn to certain task instances for which we model some details like their start and end time or the user that has executed them, in the case of user tasks.

Please note that this figure is just representative of all information that is modelled for a certain BPaaS. Much more details are captured which could not be represented in Figure 5. However, someone could explore that information by posing suitable SPARQL¹² queries over the *Semantic KB* which rely on the structure of the current modelling prototype.

¹⁰ www.bwcon.de

¹¹ <http://www.ymens.ro/en/solutii/ymens-crm>

¹² <https://www.w3.org/TR/spargl11-query/>

3.1.2 Features

This ontology exhibits various modelling features which make it suitable for supporting various types of BPaaS analysis. These features include:

- the coverage of both the type and instance level thus catering for the realisation of a models@runtime approach;
- the coverage of the organisational aspect - this enables to cover users and roles involved in customer organisations as well as link them to workflows at both the type and instance level. For instance, a role could be modelled to be used for executing a particular workflow (user) task while a user associated with this role can be recorded to execute a respective instance of this workflow task.
- the coverage of workflow dependencies, i.e., correlations between workflows and tasks and between workflow instances and task instances.
- the mapping of a certain BPaaS to a workflow as well as to the allocation decisions taken for it for traceability and best deployment analysis reasons.
- the coverage of allocation decisions/dependencies. This involves the mapping of workflow tasks to SaaS and IaaS services as well as the mapping of deployed workflow tasks to SaaS and IaaS instances.
- the coverage of the description of both IaaS and SaaS services, including important information like endpoints and location. For the latter aspect (location), we have exploited an existing ontology (the FAO¹³ geopolitical ontology¹⁴) which covers well all geographical regions until the level of countries and enables us to perform suitable querying & reasoning over them (e.g., to check which services are situated in a specific country).
- the coverage of state information for workflows and tasks which could enable also an appropriate troubleshooting (e.g., why a certain workflow task has failed).
- the coverage of the I/O parameters of tasks and their actual values upon corresponding workflow execution.
- the coverage of adaptation information which enables to derive interesting knowledge about the level of success of certain adaptation strategies of the corresponding adaptation rules triggered.
- the explicit modelling of an annotation-kind relationship to enable to map concepts like tasks and workflows to domain ontology concepts - such annotation could be quite suitable for (semantic) process mining or even service matchmaking.

All these modelling features cater for the realisation of the following types of analysis:

- *KPI evaluation*: the coverage of dependency information enables to correctly compute and correlate measurements that should be used in the computation formula of a certain KPI metric
- *KPI drill-down*: the BPaaS dependencies captured can also enable us to inspect correlations between similar or equivalent metrics in different abstraction levels for root-cause analysis purposes
- *Best deployment discovery*: by correlating deployment/allocation with measurement information we can assess how well the broker requirements have been met when a certain BPaaS was deployed in a respective cloud infrastructure and has exploited certain SaaS services. Such knowledge can then be exploited to discover the best deployments for a certain BPaaS
- *Event pattern detection*: the dependencies between the different components in the BPaaS hierarchy enables the correlation and filtering of events during event pattern detection, leading to more accurate results that are retrieved in a more rapid manner

¹³ <http://www.fao.org/>

¹⁴ <http://www.fao.org/countryprofiles/geoinfo/geopolitical/resource/>

- *Process mining*: as workflow execution history is captured by this ontology, such history can be transformed into the form of a process log on which different kinds of process mining algorithms can be operated like workflow schema (due to the recording of task execution start and end time), organisational (due to the coverage of the organisational aspect and its correlation to the dependency one), decision (due to the coverage of the concrete values for the I/O parameters of workflow tasks) and performance mining (due to the coverage of the start and end time of both workflows and workflow tasks) ones. Semantic mining can also be supported due to the coverage of annotations based on domain ontologies for, e.g., BPaaS workflows and tasks.

Thus, based on the above analysis, it can be derived that the ontology is quite rich, exhibits different modelling features and enables various types of analysis to be performed over the information structured by this ontology.

3.1.3 Implementation

The ontology has been implemented in OWL 2.0 using the Protege editor¹⁵. It is not accompanied by any parsing code as there is no need for its transformation into different models or for the production of in-memory objects for further processing. In particular, the *Harvesting Engine* just attempts to structure the information fetched via this ontology by producing the right RDF triples which are then stored in the *Semantic KB*.

3.1.4 Set-up

The ontology has been made available in the project's git repository¹⁶. A full description of the ontology can be inspected in deliverable D3.5 [1] as well as on the project wiki¹⁷.

3.1.5 Future Work

Two main directions for further work have been identified in D3.5 which mainly cover the modelling of PaaS services and their mapping to internal software components as well as the specification of the adaptation actions performed in the context of a certain BPaaS instance including their level of success. The first direction was not realised while the second direction has been partially implemented only at the modelling side. The latter means that there is no support yet for appropriately populating the *Semantic KB* with suitable information covering this aspect. This is a limitation due rather to the current implementation of the BPaaS Execution Environment prototype.

We plan to maintain and possibly expand this ontology in the near future to completely cover the requirements imposed by these two directions in order to make the ontology richer and more complete, able to cover all possible levels involved in a BPaaS hierarchy as well as to enable support for adaptation rule assessment enabling the system to better evolve its rules to the optimum. If a models@runtime approach is intended to be supported via this ontology, we might also attempt to increase the level of details for some of the concepts involved in the Evaluation Ontology.

3.2 OWL-Q KPI Extension

The evaluation of KPIs requires first their proper modelling in an extensive manner to cover all possible measurability aspects. Moreover, by considering the current trend in modelling for adopting more formal or semantic formalisms, a KPI meta-model should be semantic or ontology-based. In this sense, in the context of the previous version of this deliverable (D3.5), an ontology has been developed aimed at covering well the modelling of KPIs

¹⁵ <http://protege.stanford.edu/>

¹⁶ https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/evaluation_ontology/repository/archive.zip?ref=master

¹⁷ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Evaluation+Ontology>

and their constituent parts. To also save valuable development time, this ontology was not developed from scratch but has been built upon a prominent semantic language for non-functional service description called OWL-Q [2].

In the running period involved until the writing of this deliverable, the ontology has been evolved [3] by following some future work directions outlined for it in D3.5 [1]. This enabled the incorporation of some new but quite interesting and important features. This semantic KPI meta-model extension was well appreciated by the use-case partners which gave positive feedback, possibly related to the capability of this meta-model to completely specify all of the KPIs of their use cases. Subsequently, once this ontology was finalised, development work was conducted focusing mainly on producing suitable tools on top of this ontology. One such tool was the OWL-Q parser which enables both the parsing and writing of OWL-Q models. Such a tool can enable developers to write code on top of OWL-Q that can suit the realisation of various tasks, like the editing (in the form of textual, graphical or web-based editors), aligning and matchmaking of non-functional service models.

3.2.1 Running Example Application

The Send Invoice BPaaS has been associated with certain KPI requirements which have been specified in CAMEL¹⁸ [12]. Such requirements will be transformed into an OWL-Q specification by considering the work done at the *Harvesting Engine* (see Section 4.1). This specification, in turn, can enable the derivation of KPI queries which are evaluated against the *Semantic KB* by the *Conceptual Analytics Engine* (see Section 4.2). These KPI requirements span the following:

- The mean cycle time of the BPaaS should be less than 10 minutes
- The number of invoices sent per month should be more than 100
- The number of errors per month should be less than 10

A snapshot of the specification of the first from these KPIs based on OWL-Q is depicted in Figure 6. As it can be seen, the KPI is associated with a certain composite metric, i.e., the mean cycle time, a comparison operator as well as with the threshold imposed on this metric, thus actually representing a constraint or condition over that metric. Such a composite metric, in turn, is computed from a raw metric, i.e., the raw cycle time. This is indicated by applying the formula of this composite metric over the raw cycle time as the formula input but also by directly associating this mean cycle time metric with its component (raw cycle time) metric. Both composite and raw metrics measure certain attributes, the cycle time in our case (while throughput and reliability are measured by the metrics of the rest of the KPIs) for a specific object, which is the BPaaS workflow in our case. While the raw cycle time is computed on-demand, the mean cycle time metric is computed every hour. This is specified by mapping the mean cycle time metric to a metric context which is associated with an hourly schedule.

¹⁸ www.camel-dsl.org

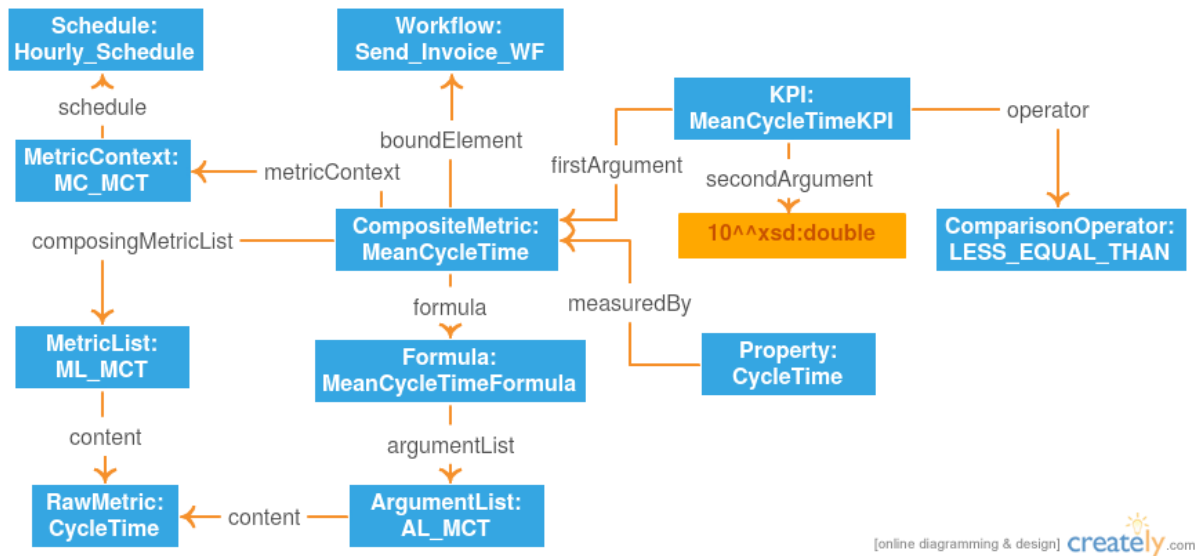


Figure 6: A snapshot of the OWL-Q model focusing on the definition of the MeanCycleTimeKPI

3.2.2 Features

We can split the features of the OWL-Q KPI extension modelling prototype into two groups: (a) features that concern the KPI extension itself; (b) features that concern the OWL-Q parser.

As the KPI extension builds upon OWL-Q, firstly some OWL-Q features are inherited, such as: (i) the coverage of multiple measurability aspects (i.e., units, value types, attributes); (ii) the flexible specification of non-functional terms; (iii) the customised level of detail (by having a minimum set of obligatory properties/relations across all concepts) but also the great coverage of each aspect covered (in contrast to the multiple properties/relations that can be specified across all concepts); (iv) capabilities for semantic validation of OWL-Q models and the production of added-value knowledge through the exploitation of semantic rules (in SWRL¹⁹). These features are then complemented with more domain-specific features which span the following (where we also indicate which feature was newly implemented or was part of the original version of this OWL-Q extension):

- Association of KPIs to goals to enable performing goal analysis and evaluate the level in which user/broker (tactical, strategic, operational) goals are satisfied (NEW);
- Exploitation of both internal and external information sources in both DB & REST-service forms to enable an even more rich and complete metric formula input specification (NEW);
- Flexible specification of KPI metric formulas with multiple opportunities for input parameter inclusion (~OLD - external information sources can be now exploited);
- Coverage of KPI parent-child relationships to enable performing root-cause analysis (OLD)
- KPI assessment knowledge capturing to enable performing different kinds of analysis, such as trend analysis and derivation, event pattern detection for KPI violations and so on (OLD);
- Coverage of all levels in the BPaaS hierarchy by mapping metric condition contexts & metrics to the BPaaS components being measured - this is the actual cross-referencing point between the Evaluation and KPI extension ontologies (OLD);
- Incorporation of domain-specific rules for domain-based validation & added-value knowledge production (OLD & NEW - NEW for the new modelling features implemented).

¹⁹ <https://www.w3.org/Submission/SWRL/>

All the above features have led to an update of the OWL-Q KPI extension meta-model which can be seen in the following figure.

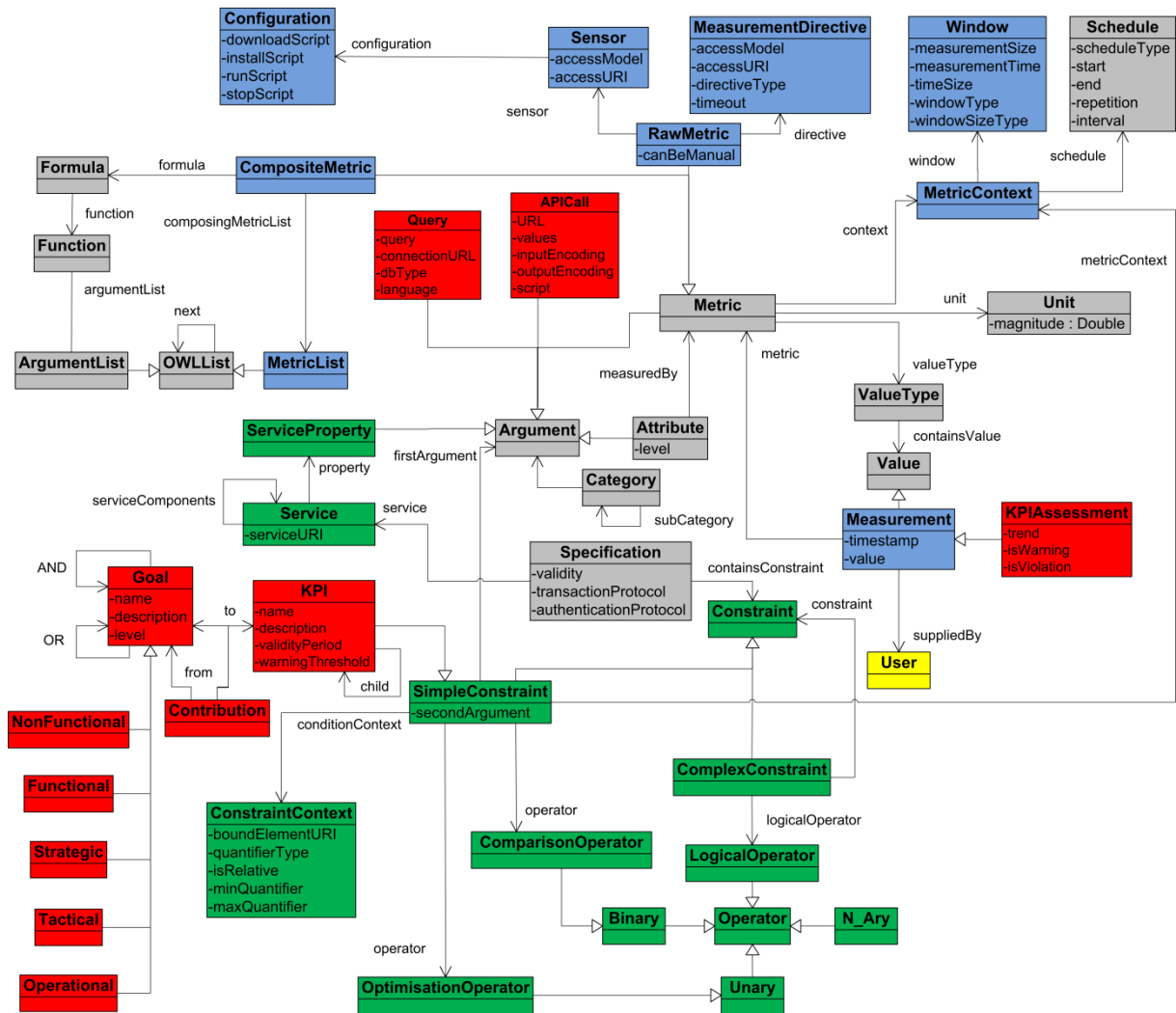


Figure 7: The update of the OWL-Q ontology in particular with respect to the KPI extension

The OWL-Q Parser exhibits, in turn, the following features:

- Ability to parse, including (syntactic, semantic & rule-based) validation, OWL-Q models in OWL into domain-level Java code;
- Ability to serialise domain-level OWL-Q code into an OWL-Q model in OWL;
- Support for various serialisation formats, including RDF/XML and Turtle, for both OWL-Q parsing and serialisation;
- Use of Java annotations in the domain-level Java code enabling the use of OWL-Q objects as I/O parameters in REST services (already done in the context of the *Conceptual Analytics Engine* - see Section 4.2).

3.2.3 Implementation

As also indicated in [7], OWL-Q is available as a set of OWL files, each mapping to the respective facet²⁰ being captured, where the KPI facet has been made a sub-facet²¹ of the specification facet. In addition, the parser for OWL-Q which covers both the OWL-Q SLA (see [7]) and the KPI extensions has been fully implemented as a maven Java project. This parser exploits the Jena framework for the loading and exporting of OWL models (in various formats) as well as the Pellet reasoner²² for their syntactic and semantic validation.

3.2.4 Set-up

The OWL-Q parser maven project is available at the project's Git repository²³. This project also includes OWL-Q's OWL files in the "input/ontology" directory. This parser can be just imported in the development environment of your choice to do some exploration and testing of the OWL-Q model parsing code. As being a maven project, the most intuitive use would be to include this component dependency in another Java component in order to build some added-value functionality which apart from pure processing could span the editing, alignment and matchmaking of OWL-Q models. In fact, the *Smart Service Discovery and Selection Tool* (see [7]) as well as the *Harvesting Engine* and *Conceptual Analytics Engine* (see Sections 4.1 and 4.2) already exploit this parsing code in order to have the ability to manipulate in-memory OWL-Q models for their own purposes.

3.2.5 Future Work

Three future work directions were identified in D3.5. These directions concerned: (a) the modelling of goals and other motivation elements; (b) the modelling of external information sources; (c) the production of (mid- or low-level) KPI models. The first two directions have been realised in the new version of this OWL-Q extension but need further extensive evaluation. The last direction has been partially realised as there was a production of OWL-Q KPI models for some of the project use cases. To this end, we plan to systematically inspect all these models and expand on them in order to produce a full-fledged KPI model which covers multiple domain-specific and domain-independent KPI terms.

The OWL-Q parser currently undergoes heavy testing and evaluation in order to be further improved. It might also be examined how to decouple the main parsing logic from the underlying semantic reasoning framework exploited. In this way, it could be possible to accompany the parser with different reasoning frameworks covering different reasoning capabilities in order to better suit the respective requirements of the broker. In this way, the broker would just have to configure the parser in order to exhibit the reasoning capability desired.

A better tailored OWL-Q editor is also planned as Protegé is quite generic and might not be so intuitive to be used by users not expert in ontology/semantic modelling. As such, we plan to produce a semantics/OWL-independent graphical editor which would cover accordingly only core & domain-specific semantics of OWL-Q in a user-intuitive way by also enabling the user to specify, if and when needed, a more customised level of detail in the specification of OWL-Q models. Such an editor would, of course, be supported by the OWL-Q parser in order to enable the import and export of the OWL-Q models edited.

²⁰ Facet means a (sub-)ontology which focuses on the coverage of a certain aspect or domain

²¹ Sub-facet means a part of a facet / ontology in this context

²² <https://github.com/stardog-union/pellet>

²³ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/owl-q/repository/archive.zip?ref=master>

4 EVALUATION ENVIRONMENT ANALYSIS PROTOTYPES

The BPaaS Evaluation Environment research prototype has been mainly evolved due to the developments in multiple of its constituent components. In fact, as also indicated in Section 1.2, all of these components reached an appropriate maturity level and most of them were integrated into the main CloudSocket implementation. These components span the harvesting of dependency and monitoring information, the KPI analysis and drill-down, the discovery of best deployment for BPaaSes, the detection of events patterns leading to KPI violations and the execution of process mining algorithms to find certain BPaaS bottlenecks. Please note that the naming of the components has been modified to refer to engines (i.e., processing and analysis components) rather than to modules. This has also been performed to have a uniform naming in the CloudSocket architecture, as most of the components of this environment have been integrated into it.

While the developments were substantial, the overall architecture of the research prototype has not been altered with the sole exceptions: (a) that the *Harvesting Engine* (previously named as *Harvester*) does not need to resort anymore to the *Meta-Model Repository* in order to obtain the definition of KPIs; (b) of the aforementioned component naming modification. In any case, for completion reasons only, this architecture is again depicted in Figure 8. To summarise, the *Hybrid Business Dashboard* is the main interaction point with users / brokers enabling them to enact analysis engines as well as to view the corresponding analysis results (with visualisation mainly drawn from information extracted from the *Meta-Model Repository*). Depending on the type of analysis needed, a different analysis engine can be invoked. Each analysis engine is offered in the form of a REST API for that reason but to also decouple the *Hybrid Business Dashboard* from the underlying analysis functionality implementation. Each analysis engine interacts with the *Semantic KB Service*, an REST-based management API for the management and retrieval of the semantic information stored in the *Semantic KB*. The interaction, in this case, is restricted mainly to the retrieval of semantic information (with the exception of the *Conceptual Analytics Engine* which is also able to store KPI measurements in the *Semantic KB*). On the other hand, the *Harvesting Engine* is a thread-based component that is executed on a timely basis, independently from the other components, with the main goal to harvest information from other CloudSocket environments, to semantic structure it and link it according to the two BPaaS evaluation modelling prototypes (see Section 3) as well as to store it in a tenant-based manner in the *Semantic KB*.

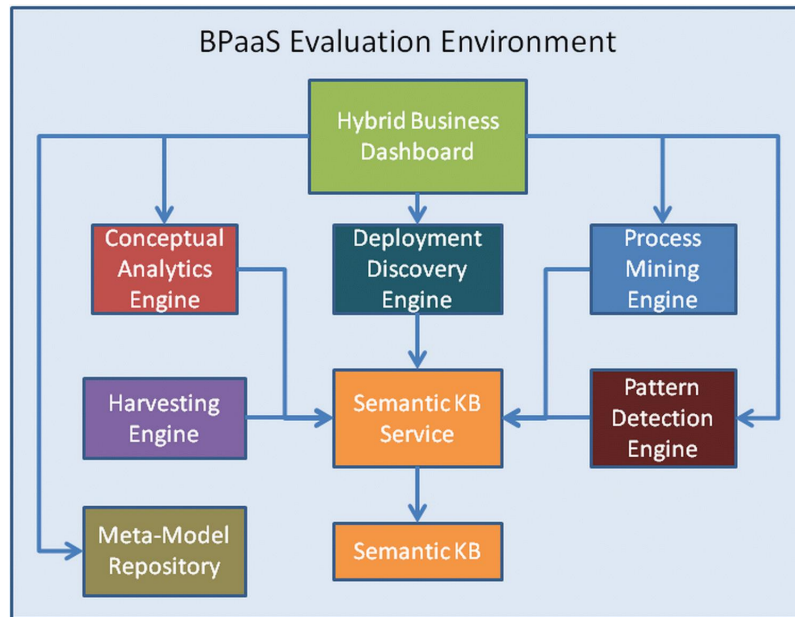


Figure 8: The overall architecture of the BPaaS Evaluation research prototype

While the overall architecture remained more or less stable, this was not the case for the components involved in it as their internal architecture was modified. Such a modification was in line with the developments and extensions performed for each component. To this end, the goal of this chapter is to introduce all the BPaaS Evaluation harvesting and analysis components with the main goal to provide an account of their current capabilities and internal architecture as well as unveil some implementation and set-up details for them. In some cases, also some future work directions are provided which explicate how these components might evolve in the near future but outside the context of this project. All this information is presented in a uniformly structured manner in separated sub-sections, each dedicated to the analysis of one component in the overall BPaaS Evaluation Environment architecture.

4.1 Harvesting Engine

The goal of the *Harvesting Engine* is to populate the *Semantic KB* in order to make it amenable for performing various types of BPaaS analysis. To this end, this component first attempts to fetch the most suitable information which mainly remains internally within the context of the CloudSocket prototype (spanning currently the *Execution Environment* and the *Repository Manager*). Then, this information is appropriately structured and semantically lifted and linked before it is finally stored in the *Semantic KB*. Semantic structuring and lifting rely on the two ontologies that have been presented in Chapter 3, i.e., the Evaluation and the KPI Extension ontology.

Two further important developments were performed for this component. First, the component has enabled the BPaaS Evaluation Environment to become multi-tenant. This was realised via the appropriate partitioning of the information collected into tenant-specific parts which are stored in tenant-specific graphs in the *Semantic KB*. The second development concerned mainly the full coverage of the monitoring aspect which was enabled via the appropriate interfacing and processing of the information collected from the *Monitoring Engine* as well as the capability to obtain and transform the KPI specifications in CAMEL into OWL-Q KPI models. This second development enabled the prototype to raise its flexibility in KPI evaluation, meaning that it is now capable of processing and analysing any KPI and not specific KPIs which were originally injected into the system. This development also paved the way for a more explorative identification of the most suitable KPI metrics as well as enabled the Evaluation Environment to be fully integrated with the rest of the CloudSocket prototype.

4.1.1 Running Example Application

As the *Harvesting Engine* is responsible for producing the respective LD specifications that conform to the two modelling prototypes and their corresponding linkage, the visualisation of the produced information for the running example has been more or less shown in Sections 3.1.1 and 3.2.1. However, in order to better highlight the linkage of the RDF information produced, we supply an example of such a linkage in Figure 9. This linkage maps to the case where we have a measurement *Meas1* which measures the raw *CycleTime* metric which is associated to the workflow instance *WI1* that maps to the deployed workflow *Send_Invoice_Dep_WF*. This deployed workflow is linked to the actual instance *Send_Invoice_Inst* of the Send Invoice BPaaS. This linkage enables us also to know to which tenant this measurement pertains as a BPaaS instance is also associated with the customer that has purchased it, something which is also shown and validated by Figure 5. As it will be indicated in Section 4.2, this linkage enables evaluation flexibility as it allows the *Conceptual Analytics Engine* to have the capability to evaluate KPIs either across all tenants or even in the context of just a certain tenant.

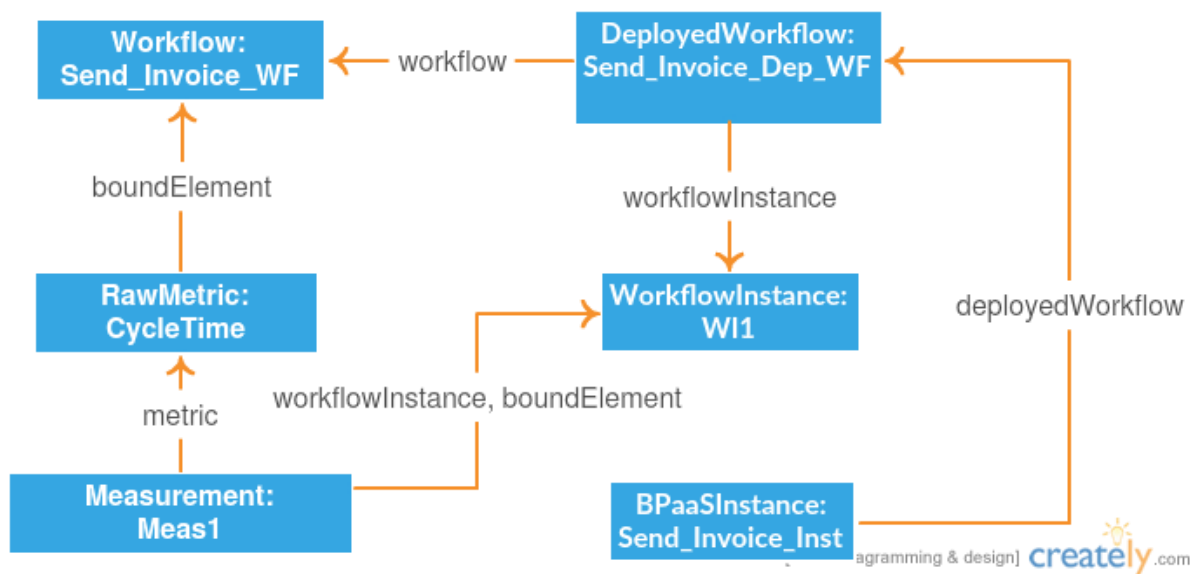


Figure 9: Example of RDF triple linkage by Harvester

4.1.2 Features

We will now provide all the features of this component in an appropriate level of detail. Apart from this more detailed presentation of the component features, this enables also new well-informed and developed implementations for it which can be interchanged with the current one. As such, the following features for the *Harvesting Engine* have been implemented:

- Tenant-aware storage of broker-specific information into tenant / broker-specific graphs
- Harvesting of information from multiple information sources within the CloudSocket prototype:
 - workflow type and instance plus partial allocation information (task-to-SaaS dependencies) and organisational information (users & roles) from the *Workflow Engine*
 - partial dependency information (internal SaaS-to-IaaS), IaaS service information & monitoring/KPI information (CAMEL-based) from the *Cloud Provider Engine*
 - measurement information from the *Monitoring Engine*
 - SaaS and software component information from the corresponding registries of the *Repository Manager*

- Structuring, semantic lifting and linkage of the information collected based on the two BPaaS evaluation modelling ontologies, the Evaluation and KPI Extension one, including the
 - CAMEL-to-OWL-Q translation capability relying only on the monitoring and requirement information stored in the CAMEL model to be transformed
- Suitable *Harvester* component configuration spanning mainly connection information to the different information sources exploited within the CloudSocket prototype

As it can be seen, the component is quite rich in features but has been realised mainly in the form of an internal software component rather than a SaaS; this is a rational choice if we consider that it supplies supporting functionality while it is also not directly enquired by the *Hybrid Business Dashboard* for BPaaS evaluation analysis purposes. We should also keep in mind that the configuration of the component is flexible which enables it to handle the migration of the information sources exploited. However, the evolution of these information sources is something that cannot be automatically addressed at this time point but is generally a difficult problem in software engineering.

4.1.3 Architecture

The internal architecture of this component was slightly modified in order to accommodate for the following change: the insertion of the *CAMEL Parser* component which supports the exploitation of certain CAMEL information for particular information harvesting purposes (such as which SaaS service is internal or external to a BPaaS and might thus require being deployed in a certain IaaS in the first case) as well as the transformation of CAMEL to OWL-Q models with current focus on the monitoring and requirement aspects. We should also note here that while the *Marketplace Extractor* exists as a component of the architecture, it has not been implemented to completely cover the organisational aspect due to some technical obstacles. However, this issue is considered as minor as the lack of some organisational information does not jeopardise the successfulness and suitability of the analysis functionality realised. In fact, some of this information can already be retrieved from the *Workflow Engine*. As such, it could be considered as a possible future work direction which could enable to perform some kind of (advanced) organisational mining over the information harvested to unveil customer or BPaaS wide trends or patterns.

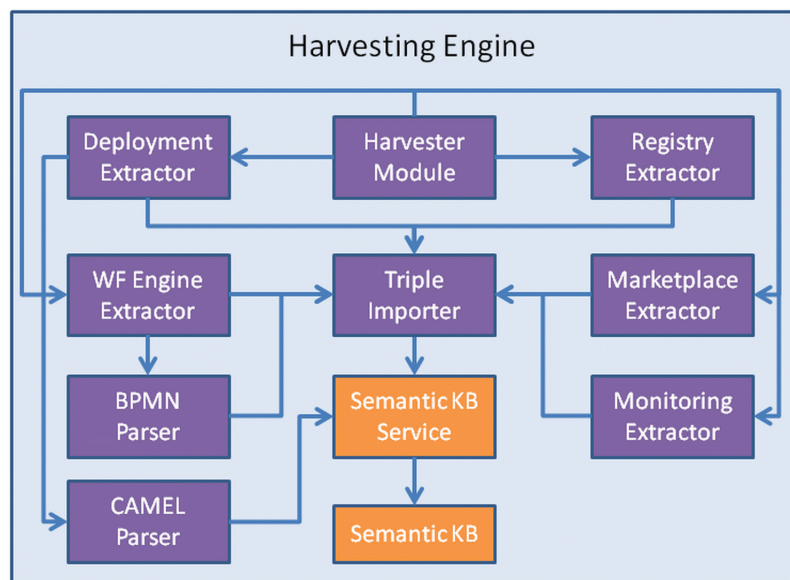


Figure 10: The architecture of the Harvesting Engine

The slightly modified architecture of the *Harvesting Engine* is depicted in Figure 10. As it can be seen, there are components which have a direct mapping to the information sources exploited. In particular, the *Deployment Extractor* is responsible for extracting relevant information from the *Cloud Provider Engine*, the *Registry Extractor* for extracting service-related information from the registries of the *Repository Manager*, the *Monitoring Extractor* for retrieving measurement information from the *Monitoring Engine*, the *Workflow Extractor* for extracting BPaaS information from the *Workflow Engine*, and the *CAMEL Parser* for extracting CAMEL information from the *CAMEL Model*. Copyright © 2017 FORTH and other members of the CloudSocket Consortium
www.cloudsocket.eu

workflow-related information from the *Workflow Engine* and the *Marketplace Extractor* for obtaining organisational-related information. Moreover, there are two utility components in this architecture: (a) the *BPMN*²⁴ *Parser* that enables to parse a BPMN workflow and to produce respective domain code that can be further processed; (b) the *CAMEL Parser* responsible for deriving suitable information from CAMEL models as well as the transformation of such information into OWL-Q models. Finally, as originally purposed, the *Harvester* is the main component which takes care of the orchestration of the rest of the components in the *Harvesting Engine*'s internal architecture in order to harvest, semantically lift and store evaluation & monitoring-related information in the Semantic KB.

4.1.4 Implementation & Set-up

The *Harvesting Engine* has been implemented as a standalone maven project and has relied on multiple libraries which span the capability to process different information types, to make information retrieval calls to information sources and to triplify the collected information. In this sense, via usual maven commands, this component can be compiled and run. It can be currently executed as a Java thread which is periodically executed in order to fetch, semantically lift and store new information, as it is being generated, without producing overwhelming load to the rest of the CloudSocket prototype system. This maven project is available at the project's git repository²⁵. The full documentation of this component can also be inspected in the project's wiki²⁶.

4.1.5 Future Work

D3.5 has identified two main directions for this component with different levels of difficulty and complexity: (a) the appropriate handling of I/O parameters and their values for BPaaS workflow tasks; (b) the capability to automatically derive the semantic annotations for BPaaS workflow tasks, in case these are not manually provided by the broker. In both cases, no actual solution was developed due to the requirement to first fix and finalise the functionality of all the main components involved in the BPaaS Evaluation Environment. However, it is our plan to pursue both directions in the near future. The non-addressing of these directions does not impact so much the analysis capabilities of the BPaaS Evaluation environment as: (a) we can safely assume that annotations to tasks are provided; (b) decision mining is not considered as a quite critical requirement but rather as a nice feature to have in the project.

4.2 Conceptual Analytics Engine

It is highly important for an organisation to have an account over the performance of its business processes (BP) as well as the right mechanisms in order to detect the root causes of BP underperformance. To this end, various KPI analysis frameworks have been proposed which differ across many criteria, such as the KPI expressiveness and the capability to support different types of KPI analysis, such as KPI drill-down. A thorough comparison of all KPI analysis frameworks can be found in D3.5.

As indicated in D3.5, we have proposed and developed a semantic approach for KPI measurement and analysis which has the following unique features: (a) it enables a more flexible way to explore the KPI metric space which is more suitable for KPI metric identification; (b) it enables users to specify KPIs in a high-level language which is not specific to a certain (storage) technology; (c) this KPI specification language is richer and more expressive than the other languages proposed; (d) this approach supports a direct method for KPI drill-down which relies on: (a) the relation between KPI metrics in a certain KPI metric hierarchy; (b) the relations between KPIs in a KPI hierarchy.

While our approach is quite sophisticated, it still had some places for further improvement. Some of these places were identified well in advance while others were derived from certain, new requirements. The former concern

²⁴ www.bpmn.org

²⁵ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/harvester/repository/archive.zip?ref=master>

²⁶ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Harvesting+Engine>

mainly the need to support multi-tenancy. Such a need caters for the situation where the CloudSocket platform operates over multiple brokers, offering its services to all of them. In this case, each broker needs to have an individual / private space via which he/she can perform KPI analysis that cannot be accessed by another broker.

Multi-tenancy is actually offered in two levels in the Evaluation Environment: (a) broker level; (b) BPaaS customer level. As the BPaaS Evaluation Environment is specifically tailored to the broker the second level is mainly used in order to produce corresponding derivations over BPaaS performance that span either single customers or all BPaaS customers. As such, there is actually no dedicated space for the BPaaS customers, driven by the fact that such customers are not actual users of the BPaaS Evaluation environment.

The new requirements were derived based on technical dependencies as well as for enabling the more optimal realisation of some features. One of these requirements involved the further separation of the broker space into two subspaces: (a) one subspace catering for the flexible exploration of the KPI metric space; (b) one subspace catering for the actual querying of KPI measurements deliberately stored in the broker space (as a result of KPI evaluation for finally selected KPI metrics) to enable a faster retrieval of KPI measurements and evaluations. This space partitioning was related to the way the KPI evaluation has been technically implemented. In particular, the injection of different kinds of measurements in a single KPI metric-testing environment would lead to inconsistencies in the evaluation of new or existing KPI metrics as the metric exploration and expansion are performed by considering which metrics map to real measurements. Thus, by both experimenting and producing values for a KPI metric would lead to a case where there is no need to expand this metric as there are measurements produced for it which would have been wrong. To this end, in this respect, non-fixed KPI metrics are evaluated based on lower-level measurements harvested from the Execution Environment, while fixed KPI metrics map directly to measurements being produced for them.

Another new requirement was related to the need to provide an account over which metrics can be used in the exploration of the KPI metric space. This is a rather fundamental requirement for supplying the right knowledge to the modeller via which KPI metric formulas can be specified. It has been realised by supplying a suitable method for retrieving this list of basic metrics.

A new requirement was also related to the need to enable a different type of KPI drill-down: instead of relying just on the direct relations between (KPI) metrics in metric tree hierarchies, relations between KPIs can also be specified which can enable to perform a more user-driven KPI drill-down. Such a feature was also realised and relied on the respective capability of the KPI modelling language (i.e., the KPI extension of OWL-Q in Section 3.2) to model such relations.

Finally, another new requirement was related to aforementioned fact that KPIs can be partially modelled in the BPaaS Design Environment and completed in the BPaaS Allocation Environment while the latter environment can also introduce new KPIs. To this end, we have realised the capability to return all the KPIs that have been specified for a certain BPaaS in order to assist in the visualisation of all these KPIs in the *Hybrid Business Dashboard*. Technically this has been made possible via the information retrieved from the *Cloud Provider Engine* which maps to the CAMEL model of a certain BPaaS. This CAMEL model, as being produced by the BPaaS Allocation Environment, already contains all the KPIs that have been defined for the BPaaS at hand.

Based on the aforementioned places for improvement, extensive development work was devoted to their realisation which has resulted in a revamp of the corresponding REST API offered. The respective analysis over this work now follows in the next sections.

4.2.1 Running Example Application

Assume that the broker desires to evaluate the mean cycle time KPI. He/she will exploit the *Hybrid Business Dashboard* to initiate the KPI evaluation functionality. In the background, this will initiate a call to the *Conceptual*

Analytics Service which will first produce the appropriate SPARQL query by considering the metric of the KPI and its context. The respective query that will be produced is depicted in Figure 11. A similar query to this one has been already analysed in D3.5. To be short, this query attempts to just gather the corresponding raw cycle time measurements into groups which are aggregated according to the AVG statistical function by also carefully selecting the appropriate objects that should be measured (i.e., instances of the BPaaS workflow). Such objects should belong to the corresponding BPaaS. We should also highlight that in case that this analysis is performed in a customer-based manner, the BPaaS workflow instances whose cycle time is being measured should have been initiated by that customer. Once the KPI is measured, it will be sent to the *Hybrid Business Dashboard* which evaluates it and depicts then its evaluation value.

```
select (AVG(?CycleTime_value) AS ?value) (MAX(?CycleTime_ts) AS ?date)
from <http://www.cloudsocket.eu/evaluation/bwcon>
where {
  ?bundle eval:id "SendInvoice";
  eval:workflow ?wf.
  ?dep_wf eval:workflow ?wf;
  eval:workflowInstance ?wf_inst.
  ?CycleTime a owl:metric:Measurement;
  owlq:metric <http://www.cloudsocket.eu/metrics/CycleTime>;
  owlq:value ?CycleTime_value;
  owlq-metric:timestamp ?CycleTime_ts;
  eval:tenant ?tenant;
  eval:wfInstance ?wf_inst;
  owlq-metric:boundElementURI ?wf_inst.
  FILTER (?CycleTime_ts >= "2017-03-04T10:39:08.703"^^xsd:dateTime &&
    ?CycleTime_ts <= "2017-03-05T10:39:08.703"^^xsd:dateTime)
}
group by (MONTH(?CycleTime_ts) as ?month, DAY(?CycleTime_ts) as ?day, HOUR(?CycleTime_ts) as ?hour)
```

Figure 11: Snippet of the SPARQL query used for the evaluation of the Mean Cycle Time KPI

4.2.2 Features

The following features relate to the current capabilities of the *Conceptual Analytics Engine*. Such features are also marked as new or old to denote whether they were realised within the second or first period of T3.3, respectively.

- a dynamic exploration of the KPI metric space via the supply of metric specifications in OWL-Q (NEW)
- period-based evaluation of KPI metrics with the additional capability to store the corresponding measurement results (OLD)
- capability to evaluate a KPI across all BPaaS customers or just for a specific customer (OLD)
- multi-tenancy (NEW)
- two different forms of KPI drill-down (OLD & NEW)
- fast querying over KPI measurement history (NEW)
- capabilities to enumerate the metrics and KPIs available for a certain BPaaS (NEW)

Based on the above feature list, it is apparent that substantial work has been conducted in delivering the new features of the *Conceptual Analytics Engine* or for extending the current ones in order to make it even more complete and sophisticated. In result, the broker has an excellent tool that can really support him/her in the appropriate performance evaluation and analysis of his/her BPaaSes that can also outline particular places for further performance optimisation.

4.2.3 Architecture

The architecture of the *Conceptual Analytics Engine* was modified to have a clear separation between the analysis and drill-down functionality. This has resulted in the updating and renaming of existing components as well as the development of new ones. The resulting final architecture is depicted in Figure 12 and highlights that the *Conceptual Analytics Engine* now comprises 6 main components: (a) the *Query Creator* which is responsible for creating an SPARQL query over a certain KPI metric; (b) the *Drill-Down Handler* which is responsible for the handling of KPI

drill-down requests by also exploiting the facilities of the *Query Creator*; (c) the *KPI Handler* which enables the uploading of KPI and metric specifications from the *Semantic Knowledge Base* and their mapping to OWL-Q domain objects which can be searchable; (d) the *Resource Accessor* responsible for the accessing of external information sources before the OWL-Q-metric-to-SPARQL transformation takes place, (e) the *SPARQL Transformer* which is exploited internally by the *Query Creator* in order to create a SPARQL query out of a certain KPI metric; (f) the *Conceptual Analytics Service* which is responsible for orchestrating the appropriate KPI analysis functionality. In particular, this REST service calls the *Query Creator* in order to obtain the corresponding query for different kinds of user requests (e.g., tenant queries, KPI evaluation, metric evaluation) and then evaluates the query over the *Semantic Knowledge Base* via the *Semantic KB Service*. It also calls the *KPI Drill-Down* component in order to initiate the drill-down of a certain KPI. Finally, it exploits the *KPI Handler* in order to answer user requests related to the retrieval of the set of metrics or KPIs that have been specified for a certain BPaaS.

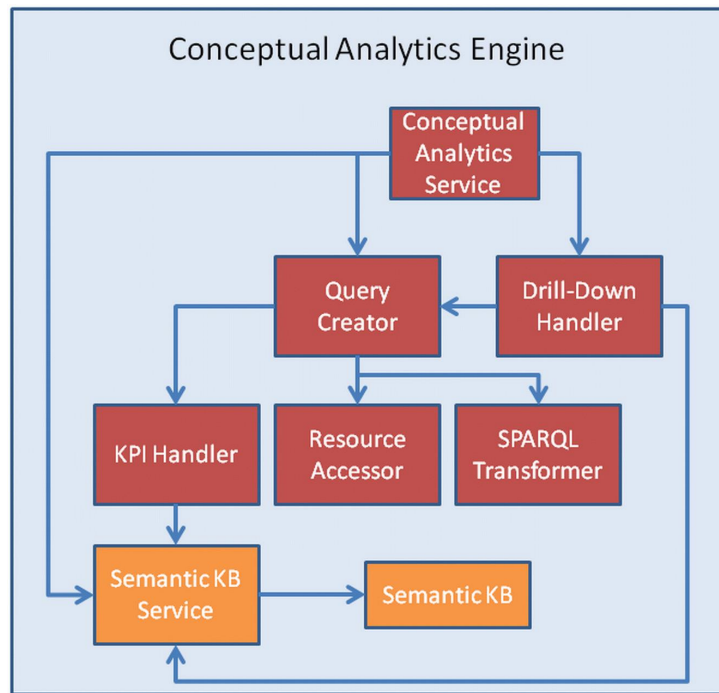


Figure 12: The architecture of the Conceptual Analytics Engine

4.2.4 Set-up

The *Conceptual Analytics Engine* component has been developed as a maven Java project which is available as open-source at the project's Git repository²⁷. This project can be compiled to produce a war file that can be deployed in a servlet container, like tomcat.

Currently, the *Conceptual Analytics Engine* runs as a service²⁸ within a tomcat container inside a VM in the UULM infrastructure. More technical details about this engine can be found on the project wiki²⁹.

²⁷ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/evaluation/repository/archive.zip?ref=master>

²⁸ <http://134.60.64.222:8080/evaluation/>

²⁹ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Conceptual+Analytics+Engine>

4.2.5 Future Work

Three main directions of work have been determined in D3.5 for this component, which include: (a) simplified metric expression via mathematical formulas; (b) the exploitation of external sources; (c) expansion of the OWL-Q-to-SPARQL transformation algorithm.

The first direction is currently followed in the implementation of the *Hybrid Business Dashboard* where after the set of resource metrics is returned by the *Conceptual Analytics Service*, a particular UI is provided in order to enable the user to build a mathematical formula based on them. Once the user builds this formula, it is transformed to an OWL-Q metric specification which is then evaluated by calling another method of the *Conceptual Analytics Service*.

The third direction was also realised due to the respective need to cover the transformation of different kinds of KPI metrics which had some interesting but technical particularities. Such particularities were related to having to address multiple component metrics that were related to the measurement of different but related objects within the BPaaS hierarchy. A very nice extension to the current solution was devised to accommodate for such more elaborate cases. In this respect, now, all of the KPI metrics mapping to the different use cases of the project can be evaluated by our BPaaS Evaluation Environment

The second direction was partially realised as there was no special need arising from the project use cases. The partial realisation relied on the detection of the need to compute the metric formula input parameter mapping to the external information source before the actual SPARQL query was constructed. In this sense, the computed value is directly inserted into the metric formula as a constant value replacing the respective (information source retrieval) element. This is a rather static way of dealing with the retrieval of external information sources. In case that the query also depends on the metric input then there is a need for a more smart mechanism which includes: (a) the modelling of special terms / symbols that map to the dynamic input in the external information source query / call; (b) the capability to call that source with this input during SPARQL query evaluation time. The second capability would require discovering a suitable way to call the source via a function. Technically this is possible but we did not have the appropriate time and incentives to realise this more involved feature.

In any case, we expect that in the near future we are going to further evaluate the delivered KPI analysis component over different BPaaSes and corresponding use cases in order to identify bugs as well as places for further improvement. Apart from its integration in the final CloudSocket implementation [8], we consider this component as an asset which should definitely be evolved and extended, possibly in the context of a new European project.

4.3 Deployment Discovery Engine

Apart from KPI analysis, another interesting kind of BPaaS evaluation concerns the capability to discover the best deployment for a certain BPaaS based on its execution history or the execution history of BPaaSes that are similar or equivalent to it. Such an evaluation kind could assist in providing interesting hints to the broker for improving the allocation of an existing BPaaS or for determining the allocation of a new BPaaS. In this respect, we conceptualised a research idea which could be realised in order to support this kind of evaluation. Such a conceptualisation relied on our previous work [13] which was able to derive the best deployments for multi-cloud applications. That work had to be extended in order to cover the additional levels involved (e.g., workflow) as well as to become semantic.

During the last period of T3.3, the aforementioned research idea was implemented based on the corresponding architecture and logic specified in D3.5. This implementation was faithful and quite straightforward thanks to our experience in both semantic and knowledge base technologies.

Before continuing with the presentation of this prototype, we need to highlight some important details to enable the reader to have a precise account of how particular challenges have led to the realisation of the current capabilities of this prototype.

The main challenge anticipated concerns the fact that each BPaaS might employ the same abstract workflow but once this workflow is concretised, then a different offering is produced with different KPIs to be guaranteed that might involve the same or different KPI metrics. Such BPaaS heterogeneity has led us to enforce some assumptions that can assist in the proper derivation of the best possible deployment for an abstract BPaaS workflow. These assumptions include: (a) only workflows which are similar or identical to the current BPaaS workflow are considered to extend the execution history that can be analysed and thus be able to derive more optimal results; (b) abstraction over KPIs and KPI metrics in order to have a uniform way of deriving when a deployment should be considered as best / optimal or not.

The first assumption enables us to also consider more BPaaSes than those exactly matching the abstract workflow of the current BPaaS at hand. This opens up more opportunities for considering a broader scope and enabling to perform a richer analysis that can lead to a much better / optimal final recommendations. Due to the lack of semantic annotations on the description of the current set of BPaaSes, we have relied on a simple but more limited task matching approach in order to infer the similarity / equivalence of the workflows involved. In particular, task matching simply relies on name matching with the main rationale that during the process of workflow concretization and enhancement for a new BPaaS design, abstract workflows can be considered as templates that constitute major units of re-use. In this sense, there is a high probability that many tasks will be maintained and have exactly the same name. This translates to the case that equivalent or similar (BPaaS) workflows usually have a great percentage of common tasks with equivalent names. A simple threshold-based rule was then involved in order to infer the workflow matching, after name-based matching of tasks has been performed. This means that when the percentage of common tasks is above that threshold, the two compared workflows are deemed as similar. On the other hand, if the task sets involved in the workflows are identical, then the two workflows are considered as equivalent³⁰.

As indicated above, the second assumption enables us to overpass the KPI (metric) heterogeneity. This has been achieved by considering that all KPI metrics are grouped into an overall QoS category. In this way, such grouping enables the formation of a tree out of which the global QoS value for a certain deployment can be derived via the transformation of the multiple KPI objectives into a single one via a weighted sum evaluation approach. In particular, each KPI metric is associated with a specific weight and utility function. The weight signifies the relevant importance of a KPI metric with respect to the rest, while the utility function enables to map the KPI metric values into a certain utility in [0.0, 1.0], thus achieving a uniform evaluation space. Such a utility function is constructed by considering the mean value of the KPI metric, the KPI threshold as well as the max and min KPI metric values among all measurements produced for the BPaaS deployment at hand across all BPaaS customers. Currently, the same weight is provided for each KPI metric but this can be changed in the near future by considering the broker preferences. In the end, the QoS of a certain deployment is computed based on the following formula: $\sum_{q=1}^Q w_q * uf_q(V_q)$, where w_q is the weight of KPI metric q which equals to $1 / Q$, Q is the number of all KPI metrics defined for the current BPaaS at hand, uf_q is the utility function for metric q and V_q is the actual mean value of the KPI metric across all BPaaS customers. By mapping each deployment to a single overall QoS value, the discovery of the best deployment just requires the ordering of all deployments based on that value and the selection of the first deployment in the ordered list.

4.3.1 Running Example Application

Suppose that we consider only the mean cycle time KPI from the KPIs given as requirements by the broker, for simplicity and demonstration reasons, and that we need to find the best deployments for the Send Invoice BPaaS.

³⁰ Please note that this is a rather loose notion of workflow equivalence as even if workflows share the same set of tasks, this does not necessarily mean that the workflows have also the same structure.

Further, suppose that 5 deployments of the BPaaS have been applied³¹ with different cycle time variations. Finally, suppose that the response time of the component services involved in the BPaaS is also considered in the analysis in the form of lower-level KPIs. The respective performance variations based on the different deployments can be seen in Table 1 below.

BPaaS	Mean Cycle Time (Minutes)	CRM Mgt.	Mean Response Time (seconds)	Invoice Mgt. (Invoice Ninja ³²)	Mean Response Time (seconds)
Send Invoice1 ³³	10	Exigo CRM ³⁴	10	Small-VM	8
Send Invoice2	9	YMENS CRM	6	Small-Medium VM	7
Send Invoice3	8	Zoho CRM ³⁵	9	High VM	4
Send Invoice4	6	Sugar CRM ³⁶	8	High-Medium VM	5
Send Invoice5	5	RedTail CRM ³⁷	7	High-Medium VM	5

Table 2: Performance variations for the BPaaS of the running example

As it can be seen, the best BPaaS deployment corresponds to respective allocation decisions that lead to medium-level support with respect to the lower-level KPI satisfaction (i.e., RedTail CRM is not optimum as YMENS CRM is better, while the same holds for High-Medium VM which leads to less optimal performance for the Invoice Ninja component with respect to the High VM). This means that there is a potential for improvement for the current BPaaS deployment. Currently, the Deployment Discovery Service will return the ordered set of deployments based on their overall utility (mapping to the sole utility of the CycleTime KPI metric in this example) in descending order, thus starting with the best deployment and finishing with the worst one. As such, the analysis is performed only at the workflow level. The non-addressing of the task-level relates to a certain technical limitation which will be addressed in our future work. This will enable the Deployment Discovery Service to report also promising combinations mapping to the best deployment for the corresponding BPaaS tasks / services (i.e., YMENS CRM for CRM Mgt. & High VM for the Invoice Ninja service component) involved. The result of the Deployment Discovery Service call, in JSON form, for this use case / running example via Swagger is depicted in Figure 13. Once the integration with the Hybrid Business Dashboard is finished, such a result will be presented to the user in a table form to supply a more suitable and user-intuitive visualisation.

³¹ Actually of the same abstract workflow mapping to this BPaaS. This means that all allocations concern a different BPaaS in the end.

³² <https://www.invoiceninja.com/>

³³ Here we actually indicate the different BPaaSes that map to the same abstract workflow. Each such BPaaS has almost the same name but is differentiated in the context of this example with a different index. Such BPaaSes in reality might have more involved names that might also reflect the type of the offering (e.g., golden, silver or bronze).

³⁴ <http://api.exigo.com/3.0/>

³⁵ <https://www.zoho.com/crm/>

³⁶ <https://www.sugarcrm.com/>

³⁷ <http://corporate.redtailtechnology.com/products/crm/>

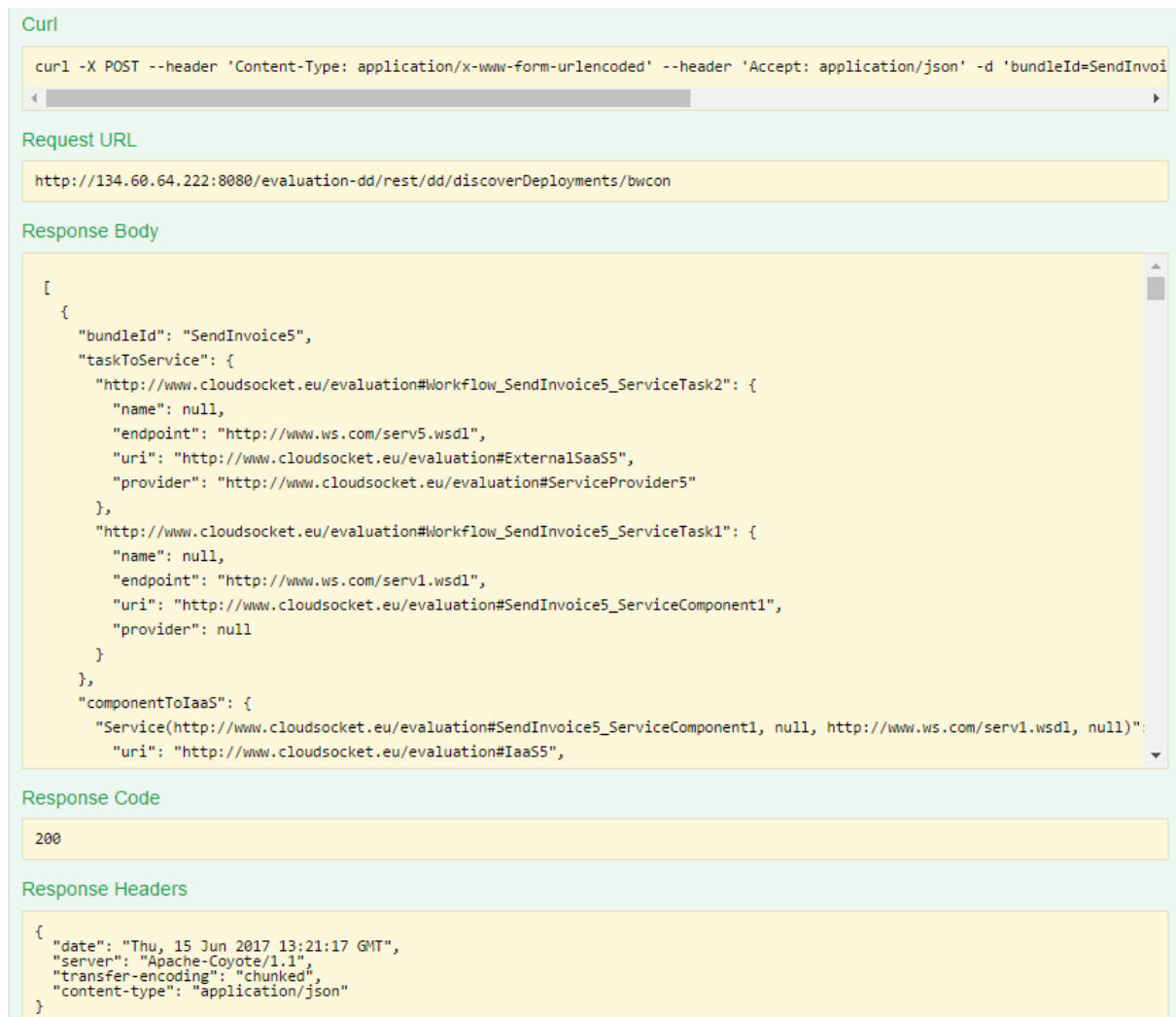


Figure 13: A snapshot of the results of the execution of Deployment Discovery Engine on the running example

4.3.2 Features

The current main features of our implementation of the *Deployment Discovery Engine* are the following:

- coverage of the workflow/BPaaS level in the analysis by considering the BPaaS deployment hierarchy across both the SaaS and IaaS levels
- possibility to consider semantics especially in resolving the resemblance of BPaaS workflows
- capability to deal with both new and old BPaaSes
- functionality exposed via REST service to decouple the user from any implementation technology
- bi-directional integration with the Smart Service Discovery and Selection tool [7] to better support the allocation of BPaaSes in a more rapid manner - this enables to exploit this component for improving the allocation of a BPaaS in the BPaaS Allocation Environment

4.3.3 Architecture

Due to the faithful implementation of the research idea sketched in D3.5, the same architecture, as defined in D3.5, has been more or less maintained with two exceptions: (a) the *Semantic Reasoner* component which has been replaced with the *Smart Service Discovery and Composition Tool* (from D3.4); (b) the introduction of a new component, called *Utility Calculator*, due to the current way of evaluating the suitability or utility of BPaaS deployments. This component exploits the aforementioned weighted approach to first derive the overall utility of a

certain deployment and then ranks all deployments in descending order based on this utility such that the top-level results map to the best deployments for the current BPaaS at hand. The corresponding modified architecture can be seen in Figure 14. The replacement of the *Semantic Reasoner* with the tool has been conducted in order to enable using more sophisticated matchmaking algorithms that can better infer the similarity or equivalence between different (BPaaS or other kinds of cloud) services. This actually enables us to have a bi-directional interaction and integration between the *Deployment Discovery Engine* and the *Smart Service Discovery and Selection Tool*. On the first hand, the latter tool is exploited in order to support the semantic matchmaking of services. On the other hand, the *Deployment Discovery Engine* can be used by the service selection algorithm in the tool in order to accelerate the service selection time by fixing parts of the optimisation problem to be solved.

Please note here that as the aforementioned tool has not been integrated into the final CloudSocket prototype in WP4, the current research prototype enables its usage but also offers the mechanism to bypass it. When tool bypassing is do configured, the semantic matching capabilities are replaced with only the name-based and threshold-based matching ones for the discovery of similar BPaaS workflows and tasks.

As originally conceived, the *Semantic Information Extractor* is responsible for fetching the appropriate information from the *Semantic KB* in the form of facts which are inserted in the *Knowledge Base (KB)*. Such facts can then be reasoned based on the set of rules carefully developed to support the discovery of best deployment for BPaaS. The orchestration of the deployment discovery functionality is still maintained at the hands of the *Deployment Discovery Orchestrator* (renamed from *Deployment Discovery Engine* in original architecture). Finally, the *Deployment Discovery Service* is the component that still exposes the current sole functionality of this engine in the form of a REST API method.

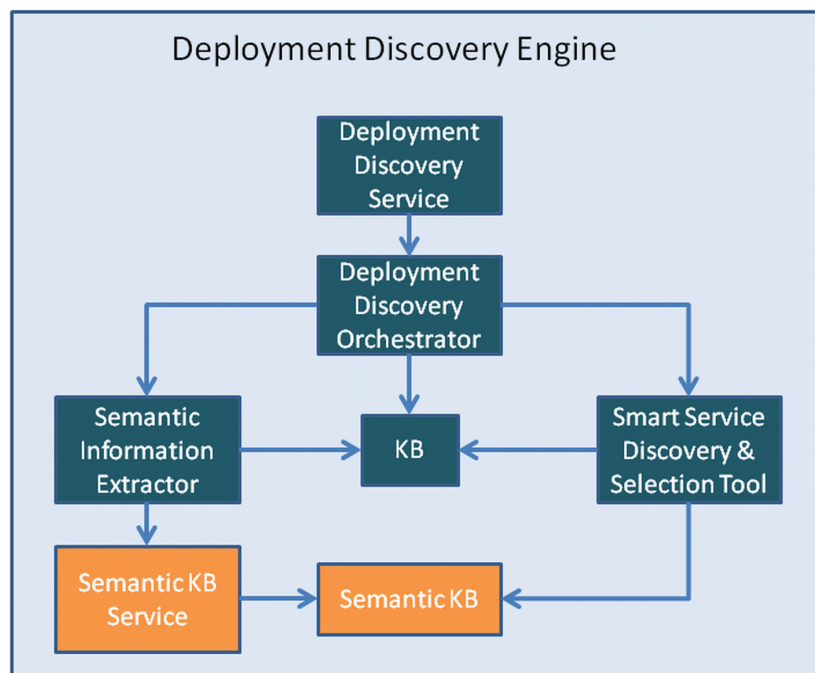


Figure 14: The architecture of the Deployment Discovery Engine

4.3.4 Set-up

The *Deployment Discovery Engine* has been realised in the form of a maven project which is available in the project's git repository³⁸. An instance of this engine runs as a service in the infrastructure of University of ULM at

³⁸ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/evaluation-dd/repository/archive.zip?ref=master>

the following URL: <http://134.60.64.222:8080/evaluation-dd/>. More technical details about this engine can be found on the project wiki³⁹

4.3.5 Future Work

In D3.5, two main research directions have been designated: (a) the coverage of the human resource aspect in the overall best deployment discovery process; (b) the exploitation of a single, uniform knowledge-based approach to support best deployment reasoning. Both directions exhibit quite involved research and technical details that might require even the creation of a project of its own. To this end, they were not actually realised. However, we would still like to remark that the second direction could make the framework more unique, uniform, and powerful so this is something we would like to pursue in the near future.

We should also highlight the need for a more semantic matching of workflows, tasks and KPI metrics as well as the coverage of the task level in the deployment discovery analysis. The former can raise the accuracy of the analysis as well as possibly enable to follow an even more focused approach in the analysis by, e.g., considering a common set of KPI metrics across all similar/equivalent BPaaSes. The matching needed can rely on our previous work on service & metric matching [2], [9], [12], provided that semantic annotations are supplied for all such elements. The task level coverage can enable the consideration of unforeseen deployments, mapping to combinations of task-level allocations, which could potentially be even better than those currently adopted and offered in the form of a BPaaS offering.

4.4 Pattern Detection Engine

A domain or rule expert might always have a hard time to derive appropriate rules that can drive the adaptive behaviour of the respective applications / business processes (BPs) at hand. This derivation process is very empirical and relies on a trial-and-error approach. Following that approach can lead to improper BP behaviour which can lead to some undesired effects like multiple SLO violations, net gains reduction and loss of market share. In this respect, domain / rule experts really require suitable supporting systems which exhibit the appropriate level of automation. Such level can involve, for instance, the automatic derivation of the required rules which are inspected and slightly modified based on the experience of the expert.

By considering the aforementioned requirement, we decided in T3.3 to work on an automatic approach to adaptation rule derivation which includes two main steps: (a) the detection of event patterns that lead to SLO/KPI violations; (b) the mapping of these event patterns to adaptation rules that can drive the adaptive behaviour of a BPaaS. This approach has relied on our previous work [14], conducted in the context of a PhD thesis, which exploited a logic-based method [15] for the detection of the event patterns and a semi-automatic method for the production of the adaptation rules mapping to these patterns based on a small set of initially-defined rules by the expert⁴⁰.

To this end, the goal of the T3.3 work was to complete the realisation of that approach by considering some requirements and restrictions that the level of BPaaS imposes. As already indicated in Chapter 1, this realisation was completed and is – although not integrated yet in the demonstration – delivered as a research item.

4.4.1 Running Example Application

Suppose that we have the following simple adaptation rules that map SLO & KPI conditions, marked with the corresponding name of the event which will be recorded in the event log if these SLO/KPI conditions are violated, to one single adaptation action that has to be executed in order to address their violation.

³⁹ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Deployment+Discovery+Engine>

⁴⁰ Here we assume that such rules are trivial to be expressed by the expert.

Rule ID	Event ID: Event: Type	Adaptation Action (ID: Name)
R1	E1: CPU > 80%: SLO	SO: Scale out Invoice Ninja SW component
R2	E2: RT > 5 sec: SLO	SU: Scale up DB component
R3	E3: RT > 10 sec: SLO	RC: Replace CRM mgt. SaaS

Table 3: Simple adaptation rules defined for the BPaaS of the running example

Further, suppose that the violation of the mean cycle time KPI maps to event E4. Finally, suppose that the following events have occurred in different time slices or traces in the BPaaS execution history.

Trace ID	Ordered Event Set
1	E1
2	E3
3	E1 E3 E4
4	E1 E3 E2 E4
5	E1 E2
6	E2

Table 4: Small trace log for the events associated with the BPaaS of the running example

The *Pattern Detection Engine* will first analyse the event log and will find out that the event pattern that is logically implied in that log and cause the KPI violation is the following: $E1\ E3 \rightarrow E4$. After discovering this pattern, the adaptation actions in rules R1 & R2 will be combined in order to produce the adaptation plan for Event E4. This will lead to generating a new rule which will have the following form: $E1\ E3 \rightarrow \text{Parallel}(\text{SO}, \text{RC})$ which indicates that when E1 & E3 events occur, then we need to perform in parallel the migration of the Invoice Ninja software component and the replacement of the CRM SaaS. Such a rule will attempt to proactively adapt the current BPaaS instance running to avoid violating the corresponding KPI requirement over the mean cycle time metric.

4.4.2 Features

The unique features of the realised work can be considered as the following:

- Almost complete automation from the moment the BPaaS is deployed until the moment the adaptation rules are derived. Only some initial, simple rules need to be defined by the expert in the form of simple event to simple adaptation action.
- Logic-based event pattern detection which is more accurate than other forms of event pattern mining, especially as it embraces a more formal approach and does not require supplying one or more thresholds which can be subjective or might require some experimental method, not easily applicable to any domain, to be applied for deriving them.
- Semi-automatic creation of complex adaptation rules able to cover more advanced adaptation scenarios. Such creation also takes into account the dependencies between the different levels in order not to produce an adaptation plan that is conflicting. In addition, it attempts to consider the best possible adaptation action for each individual event participating in the event pattern.
- Potential for semantic event pattern mining - this can come with the ability to semantically annotate as well as align events that map to equivalent non-functional metrics. As the realised approach relies on OWL-Q [2], the respective metric alignment algorithm offered for this language can be exploited to support the alignment between semantically-equivalent events.

4.4.3 Architecture

The initial architecture of this engine, analysed in D3.5, has been faithfully followed for its implementation with the sole exception of the introduction of a new component called *Transformer*. This component has the duty to transform a rule object, derived from our approach, into a respective adaptation rule description in a certain language. For the time being, this component is able to support only the transformation to CAMEL which we consider as the main pre-requisite for enabling the importing of the adaptation rules discovered in the visualisation of the respective BPaaS bundle description in the BPaaS Allocation Environment in the CloudSocket implementation. Such a visualisation will then enable the rule expert to adjust the adaptation rules according to his/her experience.

The slightly modified architecture of the *Pattern Detection Engine* is depicted in Figure 15 which shows that this engine comprises 6 components: (a) the *Pattern Detection Service* which offers the functionality to initiate the analysis functionality and present the options to either create an event pattern for a certain KPI/SLO or also the corresponding adaptation rule to remedy the violation of it; (b) the *Pattern Detection Orchestrator* which is responsible for deriving the event pattern that leads to the violation of a certain SLO / KPI while also invokes the *Information Extractor* to appropriately structure the needed information for the analysis; (c) the latter component which actually performs an extensive query over the *Semantic Knowledge Base (Semantic KB)* in order to extract and then structure the respective information needed accordingly; (d) the *Rule Derivator* which takes as input the detected event pattern and attempts to map it into an adaptation rule. For this purpose, it exploits the: (e) *Rule Base* which is exploited both as the medium via which simple adaptation rules can be fetched as well as for the storage of the adaptation rules derived; (f) finally, as already stated, the *Transformer* transforms the derived adaptation rules into a CAMEL specification.

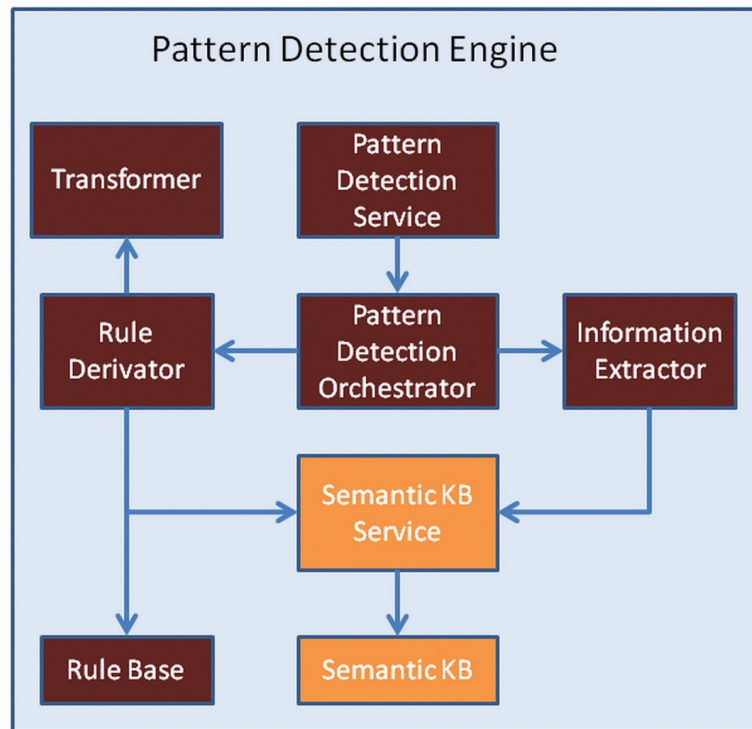


Figure 15: The architecture of the Pattern Detection Engine

4.4.4 Set-up

This component has been realised in the form of a Java maven project and is available at the project's git repository⁴¹.

4.4.5 Future Work

Although the implementation of this engine is more or less complete, we believe that two main future work directions will certainly enhance its added-value. The first direction concerns the ability to continuously evaluate the suitability of the adaptation rules derived. This could be done by a learning or a penalty/reward approach which penalises an adaptation rule when it is not successful or rewards it when it is successful. Such an approach would enable the following: (a) to interchange the adaptation plan / consequent part of the adaptation rule with another one which has been deemed more suitable according to the adaptation history; (b) to unveil those adaptation rules which need some substantial improvement which could be the focus of further analysis as well as of interference by the rule expert.

The second direction has been already mentioned. As events map to the violation of metric conditions, a semantic metric alignment approach would enable to align the semantically equivalent or similar events to each other and such alignment might lead to producing more accurate or deeper analysis results and thus to better and more advanced adaptation rules, in the end.

4.5 Process Mining Engine

Process mining is another kind of BPaaS evaluation which can lead to the generation of different types of added-value knowledge. It can enable to recreate [16] a particular BPaaS workflow based on its actual execution history. It can enable to learn some decision points [17] in order to automate the respective manual parts in the BPaaS

⁴¹ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/evaluation-pd/repository/archive.zip?ref=master>

workflow. It can also enable to derive some interesting organisation knowledge and patterns [18]. In this work, we have focused mainly on the first intended capability for two main reasons: (a) we consider it as quite important for the business of the broker and its ability to revise the BPaaS workflows deployed and run; (b) we do have the appropriate information harvested from the system to support it. On the other hand, some technical problems do not allow us to support the second type of analysis (such as the lack of knowledge about which parameters are input or output for some kinds of BPaaS workflow tasks). The third type of analysis could be supported only if the respective CloudSocket prototype was extensively used by multiple BPaaS customers with different size and organisational structures in order to then possess the right amount of information that could assist in the mining of quite interesting organisational knowledge. However, we foresee that in the near future, it would be very easy to support the latter two kinds of process mining as this would thus require just integrating suitable state-of-the-art process mining algorithms from these two kinds as well as: (a) structuring the process log for organisation mining to include organisational knowledge, something which is already supported; (b) overcoming the technical issues for the decision mining by possibly introducing some workflow engine specific extensions to mark directly the parameters as input or output for those kinds of tasks for which such knowledge cannot be easily derived.

4.5.1 Running Example Application

Suppose that there is a suitable execution history for the SendInvoice BPaaS. By invoking the *Process Mining Service*, the corresponding interactions to realise the mining functionality will be enacted. These interactions involve: (a) creating the SPARQL query in order to obtain the log for the current BPaaS at hand within a certain time period, if given by the user; (b) transforming the query results in an XES log; (c) running a specific state-of-the-art process mining algorithm called Heuristics Miner. The result of this mining algorithm will finally map to the recreation of the BPMN workflow model for the Send Invoice BPaaS as depicted in Figure 16. By comparing it with the workflow in Figure 1, i.e., the designed one, it can be seen that a certain path in the workflow is never followed which gives a modelling hint to the BPaaS designer that it might not be needed after all.

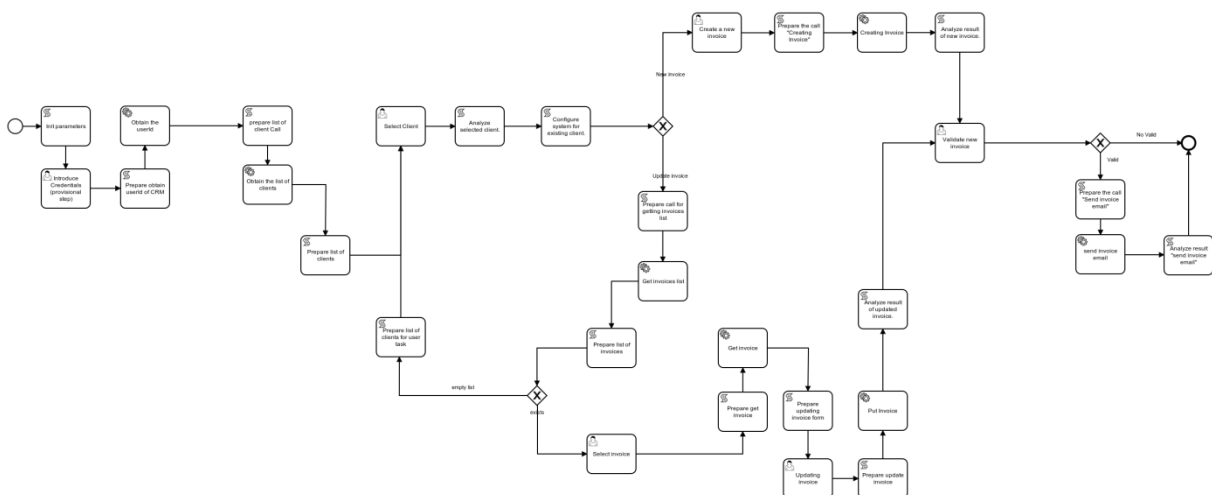


Figure 16: The workflow model mined from the execution log

4.5.2 Features

The implemented *Process Mining Engine* comes with the following interesting features:

- Support of many state-of-the-art algorithms in process mining, collected from the ProM⁴² Process Mining Framework, focusing on recreating processes from event logs conforming to the XES⁴³ standard

⁴² <http://www.promtools.org>

⁴³ www.xes-standard.org

- Support for semantic process mining via the consideration of the semantic annotations of tasks instead of simply their name in the creation of the process log - this can raise the accuracy of the mining result, provided that respective semantic annotations are in place
- Automatic log creation from the *Semantic KB* which also supports the recording of information which could be exploited for decision & organisational mining.
- Customised log creation through the supply of the analysis period as well as the actual tenant on which the analysis should be performed. By default, all tenants are considered, which makes the mining view more global at the level of the broker / BPaaS itself.
- Easy extension of mining capabilities of the system via the seamless incorporation of new process mining algorithms. Here we should confirm that such an extension is easy to realise as it just leads to reusing the existing Java-based mining algorithms from ProM, provided that the libraries on which they depend are included in the integrated code of the *Process Mining Engine*. This certainly facilitates the incorporation of different kinds of process mining algorithms in the near future to also support decision and organisational mining.
- Ability to reproduce BPMN-based processes from the (execution) log which can be visually compared with the corresponding processes designed via the *Hybrid Business Dashboard* to enable the optimisation of the BPaaS design products.

As it can be seen, our implementation already exhibits some nice features which make it quite appealing to a BPaaS broker. While only one type of mining is supported, the broker has the ability to visually check what parts of the process are actually followed and attempt to revise this process accordingly.

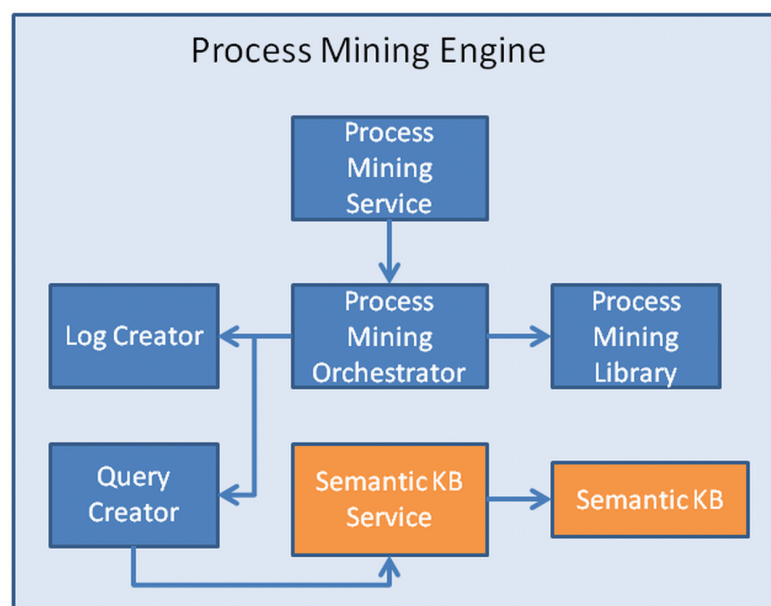


Figure 17: The architecture of the Process Mining Engine

4.5.3 Architecture

The architecture of the *Process Mining Engine* has been slightly updated with respect to the one presented in D3.5. In the final version of this engine implementation, a new component has been realised and incorporated, called *Query Creator*, which takes care of generating an SPARQL query from which the respective evaluation results can be used to produce the log on which the process mining can be performed. The updated architecture is depicted in Figure 17 which shows that the *Process Mining Engine* comprises 5 main components: (a) the *Process Mining Service* which encapsulates the process mining functionality in the form of a REST API that includes two methods which enable: (a) to dynamically figure out which process mining algorithms are supported and (b) to execute one

of them based on the semantic log extracted for the BPaaS at hand; (b) the *Query Creator*; (c) the *Log Creator* which, based on the results of the SPARQL query, takes care of producing the corresponding log to be used as input for the process mining; (d) the *Process Mining Orchestrator* which orchestrates the different components involved while also being responsible for issuing the SPARQL query and for executing the corresponding process mining algorithm selected by the broker; (e) the *Process Mining Library*, a library of process mining algorithms that are available for execution exposing also additional capabilities, such as the transformation of the process mining result (e.g., in the form of a Petri-Net) into a BPMN specification.

4.5.4 Set-up

The *Process Mining Engine* has been developed in the form of a maven Java project which is available at⁴⁴. This project can be compiled to produce a war file which can then be deployed in a servlet container. One instance of the *Process Mining Engine* already runs at a VM in the UULM infrastructure and is available at the following URL⁴⁵. More information about this engine can be found at the project wiki⁴⁶.

4.5.5 Future Work

In D3.5, four main directions for future work were outlined: (a) new organisation mining algorithms; (b) application of abstraction/taxonomy mining algorithms; (c) support for the composition of mining algorithms in the form of a workflow; (d) deployment log mining. In particular, organisation mining might lead to unveiling interesting patterns which can be followed by different types of customer organisations, while deployment log mining is a new research direction whose importance has been emphasised in the literature. It is necessary to add here as a new direction the support for decision mining which we consider as quite important for raising the automation level of BPaaS workflows. All aforementioned directions are quite interesting and might strengthen the role of the broker as a consultant, especially when transferring some specialised mining knowledge to clients. To this end, we believe that research on these topics will certainly blossom in the near future.

⁴⁴ <https://omi-gitlab.e-technik.uni-ulm.de/cloudsocket/evaluation-pm/repository/archive.zip?ref=master>

⁴⁵ <http://134.60.64.222:8080/evaluation-pm>

⁴⁶ <https://site.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Process+Mining+Engine>

5 CONCLUSION

This deliverable focused on shortly analysing the main BPaaS evaluation prototypes that have been developed in this project. In this chapter, we just summarise the contributions corresponding to these prototypes and shortly highlight what it will be done on them in the near future. Finally, we conclude by supplying a very short summary of all the main results that have been produced in WP3.

5.1 Conclusion

In this deliverable, we have presented the BPaaS Evaluation research prototype which has been also integrated as part of the corresponding environment implementation in WP4. This prototype includes 5 main components with the following capabilities:

- The *Harvesting Engine* is able to extract various information types from different information sources within the CloudSocket prototype, to semantically lift them based on two ontology prototypes, the Evaluation and the OWL-Q KPI extension ontologies, and to store them in a *Semantic Knowledge Base (Semantic KB)*. This engine enables the different types of analysis supported by the BPaaS Evaluation research prototype. It also caters for multi-tenancy due to its ability to store broker-based information into broker-specific partitions of the *Semantic KB* space.
- The *Conceptual Analytics Engine* is able to support two different types of BPaaS analysis: (a) KPI evaluation and KPI drill-down. In the case of the former ability, this engine also enables to better explore the KPI metric space by allowing users to construct KPI metric formulas and evaluate them over the *Semantic KB*. The latter ability enables to perform a certain kind of root cause analysis for detecting the main causes of KPI violations.
- The *Deployment Discovery Engine* enables to discover the best deployment(s) for a BPaaS by relying on the execution history of this BPaaS or of other BPaaS similar or equivalent to it.
- The *Process Mining Engine* enables to execute state-of-the-art process mining algorithms over a log which is built from the execution history of a certain BPaaS recorded within the *Semantic KB*. Currently, mainly process model recreation mining algorithms are exploited by this engine but this might be modified in the near future.
- The *Semantic Knowledge Base* is a semantic Triple Store encapsulated with linked data management capabilities offered by a *Semantic Knowledge Base Service*.

Apart from these components, some modelling prototypes have been developed which also provide support to the main analysis functionality of some of the aforementioned engines. These prototypes include:

- The *Evaluation Ontology* which is a semantic meta-model focusing on capturing the dependency hierarchy for a BPaaS as well as additional information aspects like the adaptation history for a BPaaS and its level of success plus organisational information.
- The semantic KPI meta-model which is an OWL-Q extension focusing on the modelling of KPIs, their dependencies and their corresponding association to organisational goals. By building on OWL-Q, this meta-model covers well all required measurability aspects. In addition, it covers the capturing of additional information, such as REST API calls or DB queries, which can be used to draw information from external sources that can be used as input in KPI metric computation formulas, as well as manual measurements along with the KPI assessment associated with them.

This deliverable also supplied an update to the *Monitoring Engine* involved in D3.4, corresponding to the BPaaS Execution research prototypes, with the main rationale that this engine is quite crucial for the proper operation of various types of analysis supported by the BPaaS Evaluation research prototype. This update mainly concerned

the ability of the *Monitoring Engine* to support additional Time Series DataBases (TSDBs) apart from the KairosDB one that also exhibit more advanced storage & analysis capabilities.

5.2 Future Work

For each section focusing on explicating the contributed BPaaS evaluation modelling and analysis prototypes, some directions have been supplied which were either followed or left for future work. By considering the BPaaS Evaluation research prototype as a whole, we consider it as a very powerful tool which can really provide great support to the broker towards optimising his/her BPaaS currently offered. Such support comes with different analysis capabilities which lead to the production of complementary added-value knowledge. In the near future, we plan to: (a) further enhance this prototype by considering the future work directions that have been designated in this deliverable; (b) to extensively evaluate it based on additional real use cases so as not only to unveil bugs but also places for further improvement. This tool enhancement and extension will possibly be realised via the participation in future European research projects. However, as this tool is also open-source, we might also attempt to obtain respective feedback and input from the open-source community which would enable us to also cover the developer side apart from the broker one plus also optimise the tool via including new features or improving the existing ones. The tool will also be internally used in FORTH and will be attempted to be used in courses along with the corresponding material to be taught based on the tight cooperation between FORTH and the University of Crete. Some future extensions of the tool could be also related to corresponding MSc or PhD thesis jointly conducted between FORTH and University of Crete.

5.3 Final BPaaS Research Conclusion

With this deliverable the BPaaS Research WP, i.e., WP3, finishes. As such, in the following, a short summary of its results is supplied. Throughout the WP, four main overall research directions were pursued, namely BPaaS design, BPaaS allocation, BPaaS execution and BPaaS evaluation.

The research blueprints mapping to these directions have been described in the respective deliverables D3.1 [19], D3.3 [9] and D3.5[1], while the actual implementation of these blueprints has been delivered in D3.2 [20], D3.4 [7] and D3.6, i.e., this deliverable. In total, more than 20 research prototypes have been implemented. In order to provide these research insights to interested projects, communities and individual researchers, all prototypes are offered for public access via the upcoming Innovation Shop (D7.7).

Out of these prototypes, the most valuable ones have been selected, in cooperation with WP4 and WP5, and have been integrated into the final CloudSocket release. Table 5 below provides a brief overview of the integrated prototypes in this final release. More details about their integration can be found in D4.6-D4.8 [4].

Integrated Research Prototypes	
Design Environment	
Name	Semantic Lifting
Summary	Align human and machine interpretation of BPaaS models
Lead	BOC, FHNW
Name	Context-Adaptive Questionnaire
Summary	User-centric framework to apply semantic lifting
Lead	FHNW
Allocation Environment	
Name	PaaS/SaaS support of CAMEL
Summary	Provide a cloud service level agnostic modelling approach for services
Lead	FORTH, UULM
Execution Environment	
Name	PaaS Orchestration
Summary	Enabling Multi-PaaS orchestration
Lead	UULM
Evaluation Environment	
Name	Evaluation Ontology
Summary	Holistic ontology for capturing BPaaS hierarchies for analysis purposes
Lead	FORTH
Name	OWL-Q KPI Extension
Summary	KPI extension of OWL-Q
Lead	FORTH
Name	Harvester
Summary	Collection, semantic-lifting & storage of execution, deployment, monitoring & registry information
Lead	FORTH

Name	Conceptual Analytics Engine
Summary	Holistic BPaaS KPI evaluation & analysis
Lead	FORTH
Name	Deployment Discovery Engine
Summary	Best BPaaS deployment derivation from BPaaS historical data
Lead	FORTH, UULM
Name	Process Mining Engine
Summary	Wrapping of process model mining algorithms into an REST-based engine with extra semantic mining capabilities
Lead	FORTH, UULM

Table 5: Summary of the integrated research prototypes in WP3

6 REFERENCES

- [1] Kyriakos Kritikos, Chrysostomos Zeginis, Daniel Seybold, and Wilfrid Utz, 'D3.5 - BPaaS Monitoring and Evaluation Blueprints', CloudSocket Project Deliverable, Dec. 2016.
- [2] K. Kritikos and D. Plexousakis, 'Semantic QoS Metric Matching', in *ECOWS*, 2006, pp. 265–274.
- [3] Kyriakos Kritikos, Dimitris Plexousakis, and Robert Woitsch, 'Towards Semantic KPI Measurement', presented at the CLOSER, Porto, Portugal, 2016.
- [4] Daniel Seybold *et al.*, 'Explanatory Notes: Final BPaaS Prototype', CloudSocket Project Deliverable D4.6-D4.8, Jun. 2017.
- [5] Robert Woitsch *et al.*, 'BPaaS Reference Models', CloudSocket Project Deliverable D5.2, Jun. 2016.
- [6] Antonio Gallo *et al.*, 'CloudSocket BPaaS Allocations', CloudSocket Project Deliverable D5.3, Jun. 2016.
- [7] Frank Griesinger, Daniel Seybold, Jörg Domaschka, Kyriakos Kritikos, Chrysostomos Zeginis, and Román Sosa Gonzalez, 'BPaaS Allocation and Execution Environment Prototypes', CloudSocket Project Deliverable D3.4, Dec. 2016.
- [8] Daniel Seybold *et al.*, 'Explanatory Notes: First BPaaS Prototype', CloudSocket Project Deliverable D4.2-D4.4, Jun. 2016.
- [9] Daniel Seybold *et al.*, 'BPaaS Allocation and Execution Environment Blueprints', CloudSocket Project Deliverable D3.3, Jun. 2016.
- [10] Andreas Bader, Oliver Kopp, and Michael Falkenthal, 'Survey and Comparison of Open Source Time Series Databases', presented at the BTW 2017 - Workshopband, 2017, pp. 249–268.
- [11] G. S. Blair, N. Bencomo, and R. B. France, 'Models@run.time', *IEEE Comput.*, vol. 42, no. 10, pp. 22–27, 2009.
- [12] A. Rossini *et al.*, 'D2.1.3 – CloudML Implementation Documentation (Final version)', PaaSage project deliverable, Oct. 2015.
- [13] K. Kritikos and D. Plexousakis, 'Mixed-Integer Programming for QoS-Based Web Service Matchmaking', *IEEE Trans Serv Comput*, vol. 2, no. 2, pp. 122–139, 2009.
- [14] C. Zeginis, K. Kritikos, and D. Plexousakis, 'Event Pattern Discovery in Multi-Cloud Service-Based Applications', *Int J Syst Serv-Oriented Eng*, vol. 5, no. 4, pp. 78–103, Oct. 2015.
- [15] A. T. H. Sim, M. Indrawan, S. Zutshi, and B. Srinivasan, 'Logic-Based Pattern Discovery', *IEEE Trans Knowl Data Eng*, vol. 22, no. 6, pp. 798–811, Jun. 2010.
- [16] W. van der Aalst, T. Weijters, and L. Maruster, 'Workflow Mining: Discovering Process Models from Event Logs', *IEEE Trans Knowl Data Eng*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.
- [17] A. Rozinat and W. M. P. van der Aalst, 'Decision Mining in Prom', in *Proceedings of the 4th International Conference on Business Process Management*, Berlin, Heidelberg, 2006, pp. 420–425.
- [18] M. Song and W. M. P. van der Aalst, 'Towards Comprehensive Support for Organizational Mining', *Decis Support Syst*, vol. 46, no. 1, pp. 300–317, Dec. 2008.
- [19] Mehmet Albayrak, Knut Hinkelmann, Kyriakos Kritikos, Sabrina Kurjakovic, Benjamin Lammel, and Robert Woitsch, 'Modelling Framework for BPaaS', CloudSocket Project Deliverable D3.1, Dec. 2015.
- [20] Knut Hinkelmann, Sabrina Kurjakovic, Benjamin Lammel, Emanuele Laurenzi, and Robert Woitsch, 'Explanatory Notes: Modelling Prototypes for BPaaS', CloudSocket Project Deliverable D3.2, Jun. 2016.

ANNEX A: LIST OF ABBREVIATIONS

List of abbreviation used into the document:

- API: Application Programming Interface
- BPaaS: Business Process as a Service
- BPEL: Business Process Execution Language
- BPMN: Business Process Model and Notation
- CAMEL: Cloud Application Execution Modelling Language
- CEP: Complex Event Processing
- DMN: Decision Model and Notation
- DSL: Domain Specific Lanugage
- IaaS: Infrastrucutre as a Service
- JVM: Java Virtual Machine
- OWL-Q: Web Onthology Language – Query Language
- QoS: Quality of Service
- PaaS: Platform as a Service
- RDBMS : Relational Database Management Systems
- REST: Representational State Transfer
- SaaS: Software as a Service
- SLA: Service Level Agreement
- SLO: Service Level Objective
- SOA: Service Oriented Architecture
- SRL: Scalability Rule Language
- TSDB: Time Series Database
- VM: Virtual Machine
- WSDL: Web Service Description Language
- WAR: Web application ARchive