# FCE4BPMN: On-demand QoS-based Optimised Process Model Execution in the Cloud

Luca Mazzola*, Patrick Kapahnke*, Philipp Waibel†, Christoph Hochreiner†, and Matthias Klusch*
*DFKI - German Research Center for Artificial Intelligence
Saarland Informatics Campus D3.2, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
Email: {Luca.Mazzola, Patrick.Kapahnke, Matthias.Klusch}@dfki.de
†Distributed Systems Group, TU Wien, Austria
Email: {p.waibel, c.hochreiner}@infosys.tuwien.ac.at

***Abstract*—One of the most important requirements for the manufacturing industry is the scalable and fault tolerant realization of their business processes. While there are already several propositions towards elastic process execution on cloud resources, compensation for faulty tasks often remains a manual task. In this paper, we present FCE4BPMN, which realizes a cloud-based execution environment, which can compensate business model executions exceptions on-demand. These compensations are performed just-in-time and ad-hoc, based on a semantically annotated BPMN model. Besides basic compensations, the optimization component is also capable of optimizing the overall process execution based on different QoS criteria. We, therefore, provide an extension to the BPMN standard as well as an execution environment to run business processes on cloud resources. Finally, we validate our approach based on a set of requirements originating from the manufacturing domain.**

***Index Terms*— *BPMN; XaaS; SemSOA; Industry 4.0; processes scalable cloud-based execution; QoS optimised process models; fault compensation re-planning; CREMA***

## I. Introduction

One of the existing issues for the practical usage of process models representation as a business engagement tool is the lack of a unified language and approach. Despite the existence of two widely adopted standard for process models (BPMN) and executable plans (WS-BPEL), no general agreement on the transformation and mapping amongst them is known. This is also due to the different approaches of the two languages: process-oriented the first, message/interface oriented the second one. Furthermore, the distinction between the process model and their instances at runtime is hard to map with an enterprise business process defined purely in term of the implementing services. For these reasons, we decided to rely only on the BPMN v2.0 as a base for our work, and we extended it to support the lacking aspects of service implementation and variable bindings and assignments. This also has three practical advantages: on one side, it simplifies the interpretation of the executable process service plans for the process designer, in respect of a BPEL-based representation. Secondly, it supports the case when the process designer would like to indicate models that are partially already implemented, meaning one or more tasks are manually preassigned with a specific service. On the last side, using extended BPMN for both process models and execution information allows for quicker reimplementation at runtime, in case a selected service is not usable (e.g: it is failing or has become unavailable).

Besides the challenges originating from the plethora of different process description and process execution plans, most approaches lack the ability to update process model execution plans at runtime. Current approaches require users to bind all the possible alternative for each service already before the start of the business process model execution, which is cumbersome for long running processes. In this case, it may be desirable to select suitable services on demand, to select the most appropriate one, which may differ according to the process model executed so far or the current service usages. This need for binding every alternative service already at startup, to maintain the flexibility during the process model execution, may result in unnecessary costs and flexibility lack. With an on demand leasing approach, as the one we are advocating for, the user delays the service selection process to a later point in time and is not required to pay for unused services, maintaining the full flexibility for exception compensation.

Additionally to the lack of flexibility during the process model execution, current process model execution engines also lack the possibility of reconfiguration whenever a service is not available. Though most process description languages and process engines support the notation of exception handling, to the best of our knowledge none of them provides the possibility to replace a faulty or unavailable service on demand. Due to the fixed service binding before the service execution, it is required to restart the whole process model execution with a newly selected service, in case of exception. This restart results in unnecessary costs, because some of the process steps need to be repeated, even if they have been correctly completed. Furthermore, in the manufacturing domain that we are taking as context, such a restart of the process execution can have heavy effects on the outcome. This is particularly relevant whenever one or more of the already executed services are not idempotent (i.e: cannot be repeated without affecting the final outcome in an unplanned way) and the semi-worked part already in the process can become unusable for the completion or a new process execution.

Here it is desirable to employ an on-demand fault compensation mechanism, which selects a new suitable service and

continues the process model execution with minimal overhead.

In the rest of the paper we will introduce FCE4BPMN (**F**lexible **C**loud **E**xecution of QoS-based Optimised Process Model for **BPMN**) by presenting a motivation scenario in section II, then we will go into the related work in Section III. The fourth section describes the approach of our solution and all the involved components of our infrastructure, whether Section V presents an initial experimental evaluation. Eventually, the conclusions are sketched in Section VI, followed by the acknowledgments to the partners of the CREMA[1] project. All the supporting materials for this paper (XSDs, examples of BPMN process model and process service plan, ontologies used) can be downloaded from the FCE4BPMN project on Sourceforge[2].

## II. Motivational Scenario

We are going to motivate the need for a flexible process model execution based on a simplified production process from the manufacturing domain [1].

The process is a very simple manufacturing one: starting from some semi-finished parts (two half non-symmetric shells and a metallic clip) it produces a key-holder.

It is composed, as shown in Fig. 1, by six sequential steps, with a parallel and an exclusive gateway to regulate the workflow. The model is realized by the following activities:

- Activate the conveyor belt, in charge of moving the working section along the production areas.
- Load the first half of the keychain body, and *parallely*, load the second half body of the keychain.
- Execute a plastic welding, generating the complete body from the two parts.
- Adding the metallic clip, through a punching mechanism.
- Checking the correctness of the assembling, through automatic finger touches.
- *If not correct*, separate the piece for manual inspection; *otherwise*, unload the semi-finished piece, to pass it to the next step in the full production (e.g., coloring) pipeline.

Every task is realizable by one or more existing services inside the architecture, which are then executable to achieve the expected manufacturing output, also thanks to the accompaining variable bindings. Despite the extreme simplicity of the process model, some aspects can already present potential issues in the execution of this process.

### *Requirements*

Based on the challenges described in the introduction and the motivation scenario, we have identified several requirements, which need to be addressed by our system design:

*a) Service Selection:* Our system design must be capable of automatically selecting concrete services for each process task based on a set of preferences provided by the process designer.

*b) Late Binding:* Although the concrete services are selected, before the process is executed, the binding of the service needs to be conducted in an on-demand manner. This allows a cost efficient process enactment, i.e., services are only obtained when they are needed.

*c) On-demand Update of the PSP:* In the case of an error, i.e., service failure, it is required to update the process service plan (PSP) on demand, to provide a feasible contingency plan to continue the process execution instead of restarting the whole process again.

*d) Abstraction of different kinds of services:* To interact with different kinds of services, e.g. software-base services or human interactions, it is required to define a common standard to easily integrate services for the process execution.

*e) Integration of PSP into BPMN:* Although the BPMN and the PSP cover different tasks in term of business processes enactment, it is required to combine them into one common format to ease the handling for the user as well as the business process execution engines.

## III. Related Work

The context of this work is multifaceted. In this section we explore the state of the art in the domain, in particular with respect to the Service Oriented Architecture (SOA) and it semantic variant used for service matching and composition; Everything as a Service (XaaS) approach; the elastic execution of process in the cloud; aspects of fault tolerance in business processes execution; and eventually, the deployment of Container based software as supporting solution for the enactment of heterogeneous services.

The key idea of semantic web services (in short: semantic services) is to enable service-based applications or intelligent agents to automatically understand what the services are doing by encoding their functional and non-functional service semantics not only in a standardized machine-readable but machine-understandable way [2]. That is achieved by describing the semantics of web service interface elements by annotating them in particular with references to concepts and rules which are formally defined in a shared ontology such as in W3C standard ontology language OWL2 or RDFS. These well-founded formal semantic annotations can then be exploited by applications and agents with appropriate formal reasoning techniques to perform, for example, automated service composition planning and high-precision service discovery.

Everything as a Service (*XaaS*) [3] is a concept common into the cloud computing infrastructure and denotes an approach where every resource is seen as a service, and the search, selection, and invocation is bound to well-defined public interfaces. Another fundamental element of XaaS is the abstraction between the concrete instantiation of the service and its (semantic) description, supporting in this way a complete decoupling of the model from the service-based plan implementing it.

Following the paradigm of Semantic Service-Oriented Architectures (SemSOA), manufacturing process models are

[1]CREMA is an EU-H2020 RIA project and the acronym stands for **C**loud-based **R**apid **E**lastic **MA**nufacturing and its website is http://www.crema-project.eu

[2]https://sourceforge.net/projects/fce4bpmn/

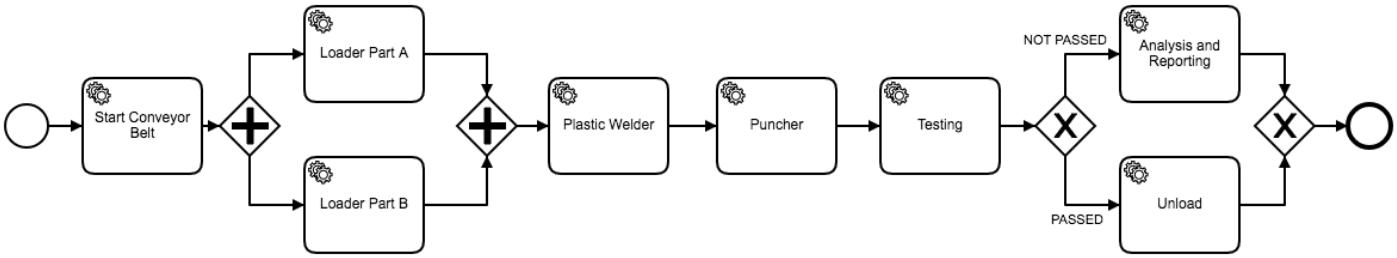


Fig. 1. Exemplary Business Process from the Manufacturing Domain

automatically implemented with semantic services by the application of appropriate techniques for semantic service discovery, selection, and composition planning.

Although the services are an established concept in computer science, the recent trend towards micro-services has triggered several innovations. One of these innovations are container based deployment techniques [4] such as Docker[3], rkt[4] or LXC[5]. These containers can be regarded as lightweight VMs that allow service developers to bundle all dependencies within one container and do not require any further configuration upon deployment. In the beginning, this approach was only considered for the micro-service domain [5], but soon they have also been applied to other domains, such as the integration of legacy services [6] or in our scenario for integrating heterogeneous services into business processes. Furthermore, containers have a better startup performance and a lower resource footprint than established virtual machines (VMs) [7], which makes them an obvious choice for an on-demand process execution environment.

To the best of our knowledge, relatively little research has been done in the field of elastic process execution [8].

ViePEP (Vienna Platform for Elastic Processes) is an eBPMS (elastic Business Process Management System) that combines the functionality of a traditional process engine with a cloud controller [9], [10]. By doing this, the platform is capable of using cloud resources, in the case of ViePEP VMs, to execute software based processes and the corresponding process tasks on them. Moreover, ViePEP optimizes the enactment of the services on the available cloud resources in a cost-efficient way without violating predefined Service Level Agreements (SLAs) [11]. Similar to our work, ViePEP uses software-based services as representative execution entities for the process tasks. However, in comparison to our approach, ViePEP does not provide an automatic service selection, as it is provided by our approach. Therefore, it can not provide the functionality described in this work, e.g., automatic service matching and composition or process optimization during the runtime in a failure case. Another difference to our proposed approach is the execution of the service on VMs instead of Containers. Despite those differences, ViePEP comes closest to our work.

[3]https://www.docker.com
[4]https://coreos.com/rkt/
[5]https://linuxcontainers.org/

Another work that uses cloud-based computational resources, in the form of VMs, in an on-demand fashion to execute process tasks is presented by Juhnke et al. [12]. Their approach is using a BPEL-based process representation. As already described, by relying on BPMN v2.0 we can simplify the interpretation of the executable process service plans.

Similar to ViePEP the works of Wei and Blake [13], Bessai et al. [14] and Cai et al. [15] are using VMs to enact business processes, respectively the process tasks of them, on cloud resources. However, also those approaches are not providing an automatic service selection, which leads again to a lack of flexibility during the process execution, especially in the case of a required reconfiguration if a service is not available.

## IV. The Approach

Our proposal is based on a series of component envisioned and partially developed in the context of a Research and Innovation Action project for cloud-based fault tolerant enhanced manufacturing process execution [1].

The starting point was the envisioning of some BPMN 2.0 extensions, to allow the models to represent the required aspect for representing and executing functionally and QoS optimal process service plans.

Then we needed a repository for the semantically annotated services, to allow our plan composer and optimizer to access all of them. For this, we designed a reduced format of OWL-S description in JSON format, that we called ServDTO. This is the service description stored in a NoSQL database. The description can represent a service (with its concrete grounding as a docker image) or simply the semantic annotation, to speed up the process model composition by pointing at it for very standard and/or repetitive tasks.

Fig. 2 graphically describes the major interactions and all the components of the proposed infrastructure. As a brief summary of the interaction, the component in charge of the plan optimal composition (ODERU) interprets the enhanced BPMN model (PM), to extract the semantic annotations for each task, and, based on the content of the service repository, computes a complete process service plan (grounding services and variable bindings and assignments), that can be executed by the runtime environment (PRU). PRU, for a task to be executed, extract the relevant service, and through the grounding information offered by the service registry, ask the service

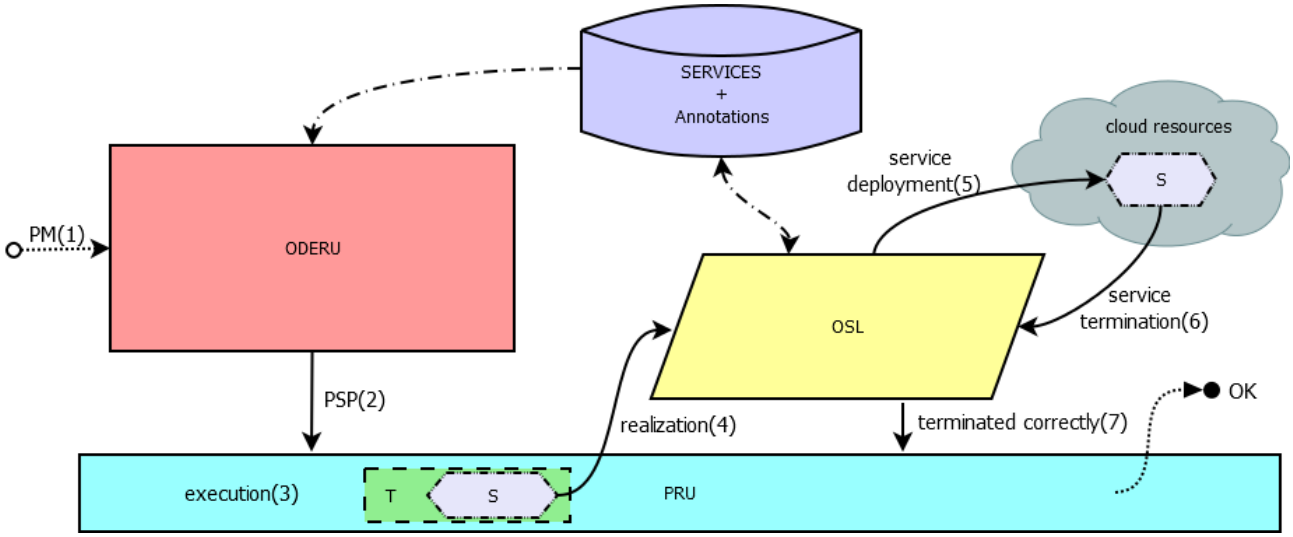


Fig. 2. The full infrastructure for the proposed solution, together with the principal interaction, in case of no exception.

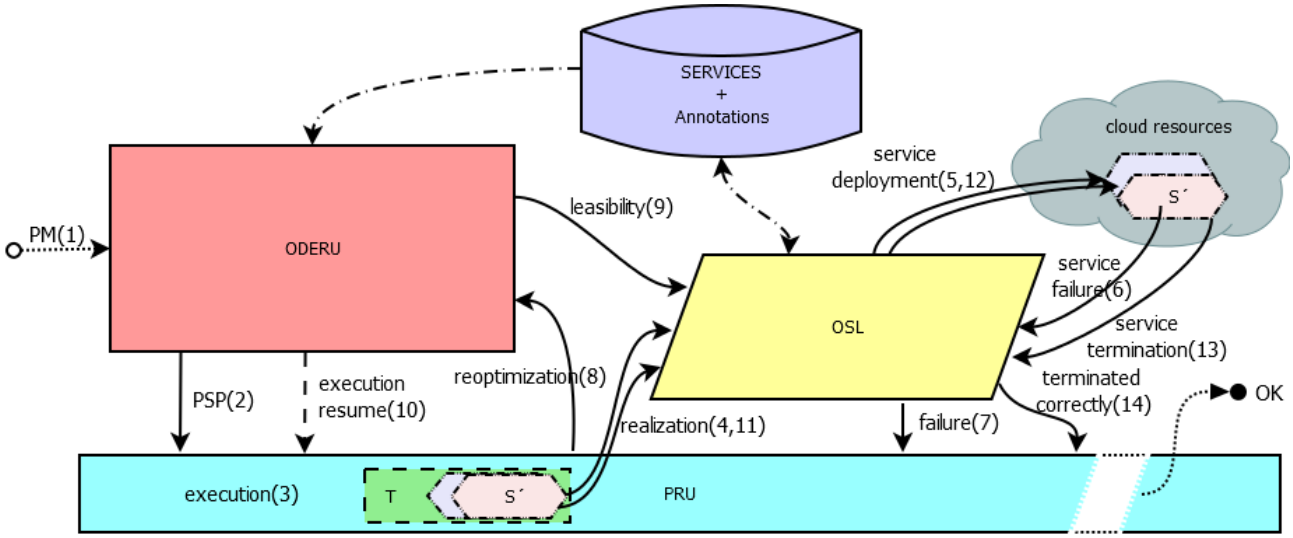


Fig. 3. The interaction in case of a service failure: the re-optimization step (8) and the execution resume (10) with the updated PSP.

leasing and realizing (OSL) to deploy and execute it into the cloud infrastructure, returning the result of this execution.

If this operation is correctly terminated for each service, the runtime environment could notify of this positive outcome (as in the Fig. 2). Conversely, if there is a failure reported back from OSL for a service deployment and/or execution (Fig. 3), PRU asks the optimization component to compute a new plan for the failed or not terminated task, taking into account also the current leasibility of the services. Once terminated this step, ODERU returns to PRU the new PSP and the runtime environment can resume the execution, using the new service(s), till the complete model is correctly executed. In the example of Fig. 3, the service $S^{'}$ substituted the failing service ($S$) in implementing the task $T$.

Eventually, each one of these components is presented in a further section, that gives the details and advancement provided by this solution, with respect to the state of the art.

### A. BPMN extensions

Semantic is taking an increasingly important role in enhancing the expressive capabilities of standard BPMN, such as in the work of [16] and [17]. On the same flow, we decided to use semantics to design our extension for services and task annotations. Equally, on the system side, the addition of semantics to BPMS to enhance business modeling standards is not a novel trend [18]. Conversely, some other work (e.g. [19]) was devoted to defining completely alternative reference architectures and frameworks for native modeling of semantic business processes.

In this section we present the extensions we designed to the BPMN v2.0 standard [20], based on the semantic annotation supported by our ontology CDM-Core [21] and its extensions.

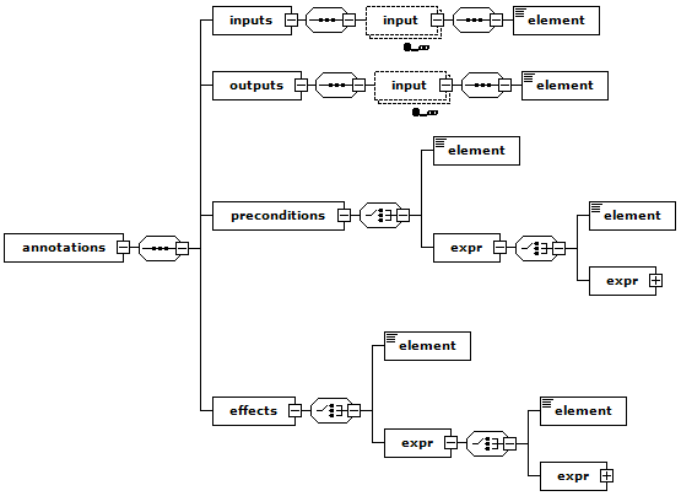


Fig. 4. XSD for the "Service Semantic Annotation".

Fig. 4 and Fig. 5 graphically present the schema for these extensions, to allow a model to bring the required information for the execution of the process inside our infrastructure. The first one (Fig. 4) includes the semantic annotation encoding of the IOPE (*I*nputs/*O*utputs/*P*reconditions/*E*ffects) annotation [22] to be added to the task, in order to allow the service selection and composition to take place automatically.

As can be noted from the Listing 1 and Fig. 4, each component of the annotation is composed by a tuple where every element addresses one field of the semantic annotation. Whether for Inputs and Outputs, the inner structure is an array of semantic elements (basically composed of a name and a semantic concept in the given ontology), the Preconditions and Effects can be more complex, as they can host (in alternative to a single semantic statement, into an element) a complex PDDL expression [23]. The PDDL expression can recursively contain another one, allowing complex logical expression to be represented. In the current examples, due to computational reasons, we developed P/E including only elements or one level limited PDDL expression (composed of a logical union of basic statements), but the tool is not structurally limited in this respect.

Regarding the extension for the process metadata, it can be explored into Listing 2 and Fig. 5. It is divided into two parts: the *optimization* and the *implementation*. The first one is devoted to all the aspects related to the *formulation* and the *results* of the non-functional constrained optimization problem for the ODERU plan optimization. In particular, into the *results* the extension allows to store the computed values of the dimensions (such as the variable assignments) to achieve the reported value of the objective function(s).

Conversely, in the *implementation* part are reported the services assignments to the model tasks, in term of grounded ("concrete") service. To maintain the linkage to the SemSOA, the pointer to the OWL-S description is also reported. To complete the information required for enacting the service in the *bindings* sub-part, each input variable of every single used service is bound to either the environment (such as one of "variable assignments" indicated by the optimal solution) or to the output of a previous service in the process plan.

With this set of information, all the required aspects of the process model implementation in term of the service-based plan becomes possible directly inside the BPMN language. Fig. 6 shows the process model of our Motivational Scenario from Sect. II in a graphical BPMN editor.

### *B. Process Service Plan composition*

The first component of the infrastructure is ODERU (**O**ptimization tool for **DE**sign and **RU**n-time), that is in charge of compose an executable functionally optimal service plan for a given process model.

This means that ODERU applies current technologies for optimal semantic selection of alternative services [24] on the functional side for annotated process tasks. The service selection is based on an exact or one step subclass plug-in match [25], to conserve the properties of the original process model. On the non-functional side, it offers support for computing exact and approximated solutions of the user given (non-)linear multi-objective COP.

The innovativeness of ODERU resides in its combined functional (semantic) and non-functional (QoS-aware) composition workflow of optimal process service plans, whether state of the art approaches for QoS-aware service composition [26], [27], [28] and semantic (process) service composition [29], [30], [31] exists in isolation. To support full enactment of the computed process service plan, ODERU also provides a possible data flow binding and the optimal service variable assignments. This functionality is provided both at design-time (when the process designer defines the model) and at run-time (when the process is executed by the runtime environment).

Another novel feature of ODERU will be its employment of RDF stream processing to react to service changes (non-functional QoS-based aspects) reported by the service registry, for example for triggering a new optimization, when a process service plan can be affected by the identified change in the stream.

### *C. Runtime environment*

The PRU (**P**rocess **RU**ntime Environment) is responsible for the execution of a process service plan, which was created by the ODERU. As a foundation, the PRU is a process engine that executes process tasks according to the order that is defined in the process, more precisely defined in the process service plan. In our approach, this process engine is furthermore capable of using the concept of flexible service selection for the execution of the process tasks. This requires an extension of the standard parsing and deployment functionality of a process engine. While standard BPMN process engines only consider the standard service definitions, a process engine for our approach has to consider also the additional PSP section shown in Fig. 5. In Fig. 7 the UI of the PRU is shown. This UI can be used to start the enactment of processes, to monitor the status of running executions, and to stop running executions.

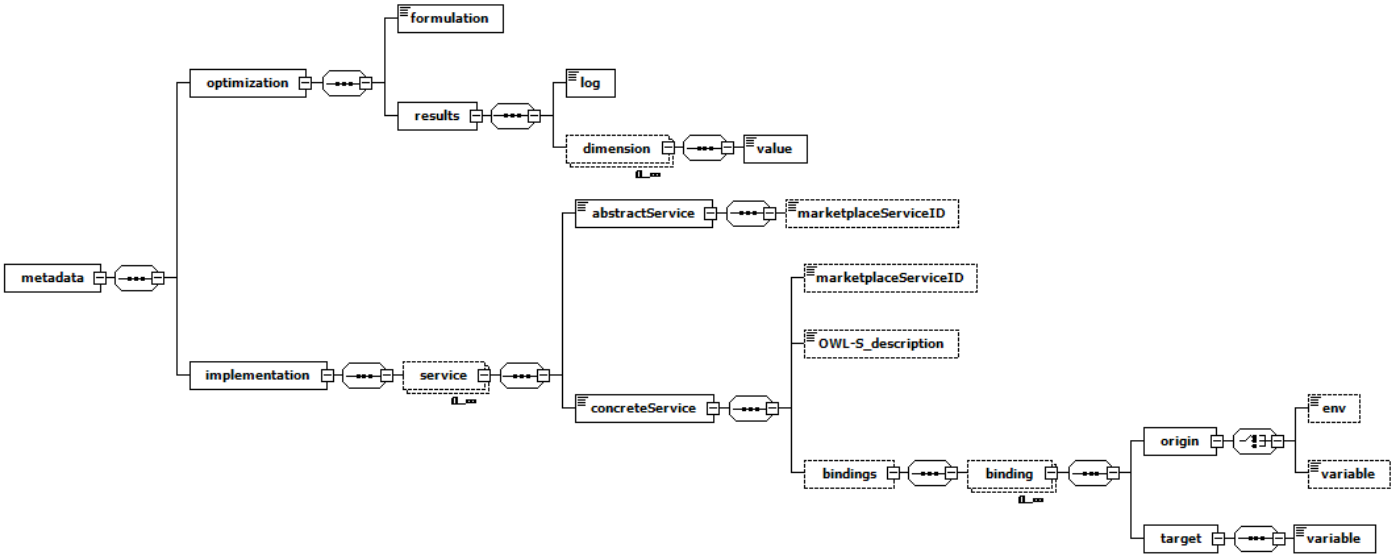


Fig. 5. XSD for the “Process Metadata”.

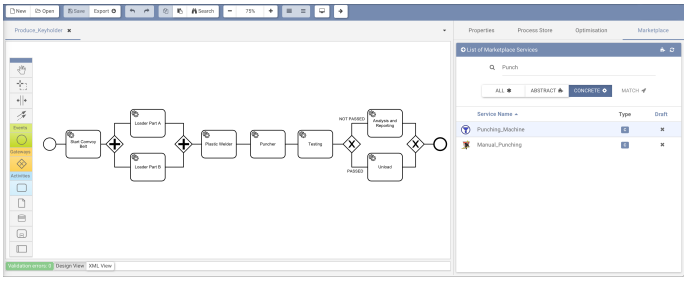

Fig. 6. BPMN Graphical Editor showing the ”Produce_Keyholder” process and on the right side possible grounded services for the ”Punch” task.

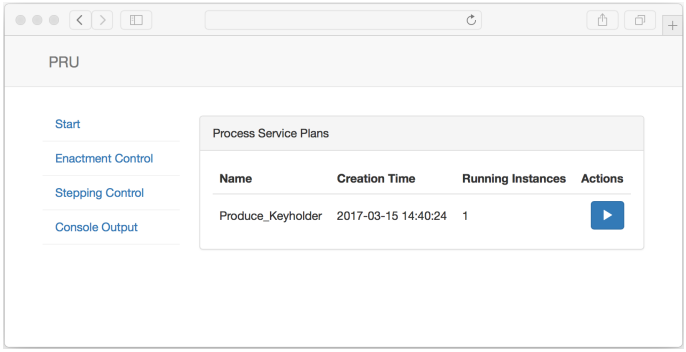


Fig. 7. Screenshot of the PRU showing a process called ”Produce_Keyholder” that has one running execution instance.

Moreover, the PRU is capable of pause running process executions, request process optimizations via the ODERU and continue the execution of the paused process with the updated process service plan from the ODERU, if an exception occurs. For example, such an exception can happen during the execution of a service, e.g., a breakdown of the hardware of a manufacturing machine, or an unavailability of a service. In the following, we discuss first the successful execution of a process (Fig. 2) and then the failure case (Fig. 3).

At the beginning of a process execution, the PRU receives the process from the ODERU (step 2 in Fig. 2). This process is then parsed, and all required information about the process tasks, the execution order, and the services is collected. Besides the standard parsing of the process defined in BPMN 2.0, our additional PSP section has to be considered. Therefore, during the parsing of the PM section, when a service definition is reached, the corresponding service definition in the PSP section (as depicted in Fig. 5) has to be mapped to the service. As described in Section IV-A this element defines for each service if it is an abstract service, i.e., without an executable software, or a concrete service, i.e., with an executable software. If it is a concrete service the defined marketplaceServiceID, i.e., the ID that is used to identify the concrete service implementation, and its corresponding variable bindings has to be stored. If it is an abstract service, an optimization of the process is requested via the ODERU to receive a concrete service that implements the defined abstract service.

Afterward, the execution of the process starts (step 3 in Fig. 2). For this, the PRU loads the first process task, according to the order defined by the model, depicted by $T$ in Fig. 2. In the case of the presented motivational scenario, this would be the ”Start Conveyor Belt” service. For each service execution, depicted by $S$ in Fig. 2, the PRU requests the execution of the service via the OSL (step 4). This request to the OSL contains all required information, i.e., which service to execute and the required input parameters, to execute the service. After the request is sent, the PRU waits for a response from the OSL, i.e., if the service execution was successful or not, if the execution ordering is sequentially, or starts an execution of a second service if the model defines a parallel execution of the services. For example, in our motivation scenario, the

services ”Load Part A” and ”Load Part B” are executed in parallel. After the PRU receives a response from the OSL (step 7) the next service is executed according to the model. After all process tasks have finished, the execution of the process is completed, and a responsible operator is informed about this fact.

However, if the service execution fails the OSL informs the PRU about the exception (step 7 in Fig. 3). When the PRU receives such a failure message the execution of the process is paused, and the current execution state is persisted. Subsequently, an optimization of the process, via the ODERU component, is triggered (step 8). This request contains all information about the occurred exception as well as all information about the already executed services of the current process execution, inform of a process execution log. After the optimization is done, the ODERU informs the PRU about the optimized process (step 10). The PRU then continues the paused process, beginning with the process task that contained the failed service. In Fig. 3 the new service is depicted by *S'*. The continued execution then requests the execution of *S'* via the OSL, which then executes it and returns the result back to the PRU (step 11 - 13). Again, after all process tasks are completed, the execution of the process is done and an operator is informed.

### D. Service lease and release

The OSL (**O**n-demand **S**ervice **L**easing and Releasing component) is responsible for the enactment of services on-demand on cloud resources to execute business processes. In the manufacturing domain, the kinds of services range from traditional software services, e.g., analysis of values, over real world services, e.g., welding parts, to human-based services, e.g., load or unload parts. To combine these services, we have designed *Proxy Service Wrappers*, which provide a uniform representation of different kinds of services, so that they can be used by processes.

*Proxy Service Wrapper:* The basic foundation of a Proxy Service Wrapper (PSW) is a set of requirements, which need to be implemented by services to be integrated. The first requirement is that each service exhibits two different endpoints. The first endpoint is the *Availability* endpoint, which checks the availability of the actual service. While it is trivial to identify the availability of software services because they have no external dependencies and can be easily replicated, it is harder to evaluate the availability of real world services, e.g. welding robots, or humans. Real world services, may be already occupied for other process enactments or the machine can also be broken. The same applies for human-based services because the human service can be either occupied or is not available, e.g., during breaks.

The second endpoint is the *Start* endpoint, which accepts input parameters in the form of a JSON object and starts the execution of a service. This endpoint can be only executed if the service is available which is ensured by a preceding call to the availability endpoint. As soon as the start endpoint is triggered, the PSW starts the operation for a software-based service, triggers a real world interaction, e.g., starts a welding process, or signals a human that he can start working on a specific task.

The second requirement is, that the service needs to register to an endpoint of the OSL to report when the service has finished its execution, e.g., a software calculation or a welding operation is finished or a human indicated that the operation is done, or whenever an error occurred, so that the OSL can inform the PRU to start the compensation mechanism.

The last requirement is that all services need to be represented within a common container format to ease the deployment on cloud resources. For our system design, we have chosen the Docker Image format due to its widespread distribution and the ability to package all kinds of external resources within the image.

Given these three requirements, our system design is capable to easily integrate all kinds of services from the manufacturing domain.

*OSL System Design:* The main task of the OSL is to instantiate services, which are represented by PSWs, on cloud resources. Therefore it needs to be capable of obtaining both the PSW images as well as cloud resources to run the services on these cloud resources. Furthermore, it needs to run these PSWs to run the actual services, interact with the endpoints of the PSW to handle all interactions and communicate with both the ODERU and the PRU to realize the process enactment.

In Fig. 8 the UI of the OSL is shown. This UI can be used to monitor the status of deployed and running PSWs and to stop their execution.

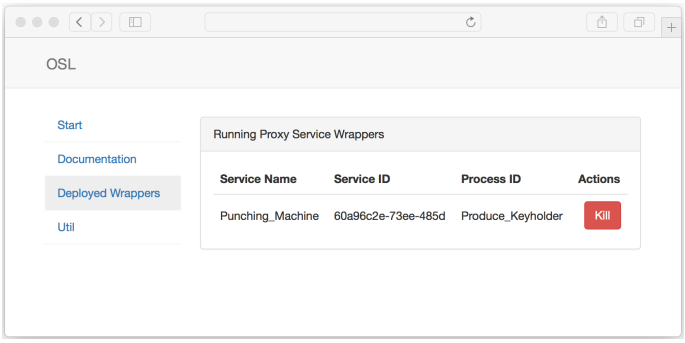


Fig. 8. Screenshot of the OSL showing a deployed and running PSW called ”Punching_Machine”, which is used in the ”Produce_Keyholder” process.

*OSL Interaction:* First, we are going the sketch the order of events for a successful service execution. At the beginning, the OSL receives a realization requests (step 4 in Fig. 2). Based on the realization request, the OSL obtains the PSW image from the service repository and obtains the cloud resources to run the service. After these preliminary steps are finished, the PSW is deployed on the cloud resources (step 5 in Fig. 2). This deployment not only contains the instantiation of the PSW but also an availability check (which we assume in this scenario as available) and starts the execution of the service. As soon as the service is finished, the service notifies the OSL (step 6 in Fig. 2), which in turn also notifies the PRU (step 7 in Fig. 2)

of a successful service execution. After the PRU is notified, the OSL un-deploys the PSW to release the cloud resources again.

In the case of a service execution failure, the first steps are the same as for the successful scenario (step 4 and 5 in Fig. 3). The differences begin, when the actual service execution fails, e.g., the real world machines breaks down. In this case, the PSW detects the machine failure and reports a service failure to the OSL (step 6 in Fig. 3) who propagates this failure to the PRU (step 7 in Fig. 3) for further handling and un-deploys the PSW of the faulty service. This further handling is then conducted by the ODERU and the result of the handling is a new service selection, which is forwarded to the OSL by the PRU (step 11 in Fig. 3). This time, a different service has been selected, which executes the service without any failure. As soon as the service finishes its execution, the PSW reports this to the OSL, which continues its operation in the same manner as for the successful scenario and waits for new realization requests of the PRU.

## V. Requirements Validation

As a proof of concept, in the simple scenario of Fig. 1, we hypothesize that the task "Puncher" was grounded to an automatic robot arm, implementing the service of punching the clip. We also imagine that during the execution of the plan the robot arm becomes suddenly unavailable. Thanks to our infrastructure, a human operator can substitute the original robot arm, by engaging in the punching activity. Under the assumption that the operator is described with enough precision as a service, ODERU can match it automatically to the task "Puncher", once the original plan fails. For a better analysis, we go through the requirements defined in Sect. II and explore how the infrastructure tackles and solves them.

*a) Service Selection:* Given the IOPE semantic annotations of task and service, FCE4BPMN can match and select automatically grounded services to compose a valid PSP, including the variable bindings and assignments.

*b) Late Binding:* Once the prepared PSP has to be executed in the PRU, a late just-in-time service grounding enactment through the functionalities of the component OSL is performed.

*c) On-demand Update of the PSP:* As demonstrated by the scenario, the update of the process service plan as consequence of a service enactment failure is performed automatically. This implements a need-based contingency planning that allows execution to be continued, minimizing (to the level of a single task) the possible negative effects of non-idempotent failed service execution.

*d) Abstraction of different kinds of services:* This is due to the SOA approach, as the separation between the grounding level and the (semantic) description of services supports this abstraction on its essential form. Our solution extends it as proposes an intermediate level of an executable wrapper, used to unify the offered functions (in term of REST interfaces) regardless of the practical implementation details.

*e) Integration of PSP into BPMN:* Despite the existence of specific languages for the service plan execution (such as the Business Process Execution Language BPEL [32]) and many approached to (semi-)automatically translate BPMN into BPEL models (such as the BPMN2BPEL[6] google project), the added values of merging into the process modeling language the execution aspects are twofold. On one side, it allows maintaining in a single place the full set of model and grounding-related information. On the other side, it supports the separation of the model from the instance level, managed only at runtime. The instantiating environment PRU is then allowed to decide based on the values of variables (including also environment related ones, if needed) the actual path followed on gateways, not requiring the identification of the process instance before the service grounding identification for task assignment.

As it can be noted, our infrastructure FCE4BPMN respects all the identified requirements. Obviously, many and different requirements can emerge from other use cases, and we will work towards the identification and coverage of them in our future work.

Despite the presented added values, we are aware of some limitations our approach intrinsically presents. The most relevant one relates to the expressive capabilities of the optimization component (ODERU) and its approach for smart process plans creation. For the service plan a smarter QoS-based optimization [33] is currently under analysis. It will also consider the inter-living of functional and non-functional aspects, to improve the created PSP. Another aspect focuses on the support of generic QoS measures, as defined by services annotations. This will allow the coverage of dimensions specific to the context, such as in case of addition of a new scenario in other manufacturing domains.

## VI. Conclusion

In this paper, we presented an approach for process service plan execution in the cloud called FCE4BPMN. It supports scalability thanks to the semSOA approach and the usage of cloud resources in the service enactment.

The infrastructure was described in details, explaining the role and interactions amongst the designed components, also using a motivational scenario as support to present some requirements and explore how our approach tackle them.

On the top of the single parts described, the most important contribution of this work is the combination of the following aspects:

- some well-formed extensions of the BPMN standard, for encoding all the execution related aspects (i.e., semantically enhanced annotation of tasks, services-based implementation, variable bindings and assignments, QoS constraints for optimization, a result of a non-functional optimization).
- a new approach for functional composition, non-functional (QoS based) optimization and execution of

[6]https://code.google.com/archive/p/bpmn2bpel/

process models, using the defined extensions of BPMN language.
- a runtime environment able to execute the "enhanced" BPMN-encoded service plans, and react to unexpected failure by providing a process execution log used for the re-optimization.
- a lease and release mechanism supporting the deployment of Docker images into the cloud for elastic scaling, based on process requirements.

On the top of the testing with some simple manufacturing-based examples, we provided an example scenario that demonstrated the coverage of the elicited requirements.

## ACKNOWLEDGMENT

This work was partially financed by the European Commission H2020 RIA project called CREMA, under the agreement 637066. The authors would like to thank all the project partners for the comments and contributions to the ideas behind the realized solutions.

## REFERENCES

[1] S. Schulte, P. Hoenisch, C. Hochreiner, S. Dustdar, M. Klusch, and D. Schuller, "Towards process support for cloud manufacturing," in *18th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2014, pp. 142–149.

[2] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic web services," *IEEE intelligent systems*, vol. 16, no. 2, pp. 46–53, 2001.

[3] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (xaas) on the cloud: origins, current and future trends," in *IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 621–628.

[4] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[5] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, 2015.

[6] A. Slominski, V. Muthusamy, and R. Khalaf, "Building a multi-tenant cloud service from legacy code with docker containers," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2015, pp. 394–396.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2015, pp. 171–172.

[8] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic business process management: state of the art and open challenges for bpm in the cloud," *Future Generation Computer Systems (FGCS)*, vol. 46, pp. 36–50, 2015.

[9] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Introducing the Vienna Platform for Elastic Processes," in *Performance Assessment and Auditing in Service Computing Workshop at 10th International Conference on Service Oriented Computing*, ser. LNCS, vol. 7759, 2013, pp. 179–190.

[10] P. Hoenisch, C. Hochreiner, D. Schuller, S. Schulte, J. Mendling, and S. Dustdar, "Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds," in *8th International Conference on Cloud Computing*, 2015, pp. 17–24.

[11] P. Hoenisch, D. Schuller, S. Schulte, C. Hochreiner, and S. Dustdar, "Optimization of complex elastic processes," *Transactions on Services Computing*, vol. 9, no. 5, pp. 700–713, 2016.

[12] E. Juhnke, T. Dörnemann, D. Bock, and B. Freisleben, "Multi-objective Scheduling of BPEL Workflows in Geographically Distributed Clouds," in *4th International Conference on Cloud Computing*, 2011, pp. 412–419.

[13] Y. Wei and M. B. Blake, "Proactive virtualized resource management for service workflows in the cloud," *Computing*, vol. 96, no. 7, pp. 1–16, 2014.

[14] K. Bessai, S. Youcef, A. Oulamara, and C. Godart, "Bi-criteria strategies for business processes scheduling in cloud environments with fairness metrics," in *7th International Conference on Research Challenges in Information Science*, 2013, pp. 1–10.

[15] Z. Cai, X. Li, and J. N. Gupta, "Critical Path-Based Iterative Heuristic for Workflow Scheduling in Utility and Cloud Computing," in *11th International Conference on Service Oriented Computing*, ser. LNCS, vol. 8274, 2013, pp. 207–221.

[16] W. Abramowicz, A. Filipowska, M. Kaczmarek, and T. Kaczmarek, "Semantically enhanced business process modeling notation," in *Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications*. IGI Global, 2012, pp. 259–275.

[17] D. Karastoyanova, T. van Lessen, F. Leymann, Z. Ma, J. Nitzche, and B. Wetzstein, "Semantic business process management: Applying ontologies in bpm," in *Handbook of Research on Business Process Modeling*. IGI Global, 2009, pp. 299–317.

[18] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel, "Semantic business process management: A vision towards using semantic web services for business process management," in *IEEE International Conference on e-Business Engineering (ICEBE) 2005*. IEEE, 2005, pp. 535–540.

[19] M. Dimitrov, A. Simov, S. Stein, and M. Konstantinov, "A bpmn based semantic business process modelling environment," in *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007)*, vol. 251, 2007, pp. 1613–0073.

[20] T. Allweyer, *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016.

[21] L. Mazzola, P. Kapahnke, M. Vujic, and M. Klusch, "Cdm-core: A manufacturing domain ontology in owl2 for production and maintenance," in *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD,*, 2016, pp. 136–143.

[22] J. Cardoso, "Discovering semantic web services with and without a common ontology commitment," in *IEEE Services Computing Workshops(SCW'06)*. IEEE, 2006, pp. 183–190.

[23] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[24] M. Klusch, P. Kapahnke, S. Schulte, F. Lecue, and A. Bernstein, "Semantic web service search: a brief survey," *KI-Künstliche Intelligenz*, vol. 30, no. 2, pp. 139–147, 2016.

[25] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An integrated semantic web service discovery and composition framework," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 537–550, 2016.

[26] A. Strunk, "Qos-aware service composition: A survey," in *2010 IEEE 8th European Conference on Web Services (ECOWS)*. IEEE, 2010, pp. 67–74.

[27] D. Schuller, A. Polyvyanyy, L. García-Bañuelos, and S. Schulte, "Optimization of complex qos-aware service compositions," in *International Conference on Service-Oriented Computing*. Springer, 2011, pp. 452–466.

[28] G. Zou, Q. Lu, Y. Chen, R. Huang, Y. Xu, and Y. Xiang, "Qos-aware dynamic composition of web services using numerical temporal planning," *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 18–31, 2014.

[29] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An integrated semantic web service discovery and composition framework," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 537–550, 2016.

[30] M. Klusch and A. Gerber, "Fast composition planning of owl-s services and application," in *ECOWS'06. 4th European Conference on Web Services, 2006*. IEEE, 2006, pp. 181–190.

[31] M. Born, J. Hoffmann, T. Kaczmarek, M. Kowalkiewicz, I. Markovic, J. Scicluna, I. Weber, and X. Zhou, "Semantic annotation and composition of business processes with maestro for bpmn," in *European Semantic Web Conference*. Springer, 2008, pp. 772–776.

[32] OASIS. (2007) Web services business process execution language version 2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[33] F. Baligand, N. Rivierre, and T. Ledoux, "A declarative approach for qos-aware web service compositions," in *International Conference on Service-Oriented Computing*. Springer, 2007, pp. 422–428.

Listing 1. Snipplet of BPMN v2.0 extension for IOPE semantic annotation of a task. In **green** the semantic concepts and properties used for the element defintions.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="http://www.omg.
    org/spec/BPMN/20100524/MODEL" xmlns:bpmndi=
    "http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:di="http://www.omg.org/spec/DD
    /20100524/DI" xmlns:dc="http://www.omg.org/
    spec/DD/20100524/DC" xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance" xmlns:
    camunda="http://camunda.org/schema/1.0/bpmn
    " xmlns:crema="http://crema.project.eu" id=
    "Definitions_1" targetNamespace="http://
    bpmn.io/schema/bpmn">
  <bpmn:process id="Process_1" isExecutable="
      true">
    <bpmn:startEvent id="StartEvent_1">
      <bpmn:outgoing>SequenceFlow_0jwbjxh</bpmn
          :outgoing>
    </bpmn:startEvent>
    ...
    <bpmn:sequenceFlow id="SequenceFlow_0byzekd
        " sourceRef="ExclusiveGateway_0ezz2gb"
        targetRef="ServiceTask_00qwy4n" />
    <bpmn:serviceTask id="ServiceTask_1m1ppqz"
        name="Puncher" camunda:type="external"
        camunda:topic="CremaServiceExecution">
      <bpmn:extensionElements>
        <crema:annotations>
          <crema:inputs>
            <crema:input>
              <crema:element name=Be1> http://
    www.crema-project.eu/ExtensionA.owl#Belt</
    crema:element>
            </crema:input>
            <crema:input>
              <crema:element name=Ke1> http://
    www.owl-ontologies.com/mason.owl#Kernel</
    crema:element>
            </crema:input>
            <crema:input>
              <crema:element name=Cl1> http://
    www.crema-project.eu/ExtensionA.owl#Clip</
    crema:element>
            </crema:input>
          </crema:inputs>
          <crema:outputs>
            <crema:output>
              <crema:element name=Se1> http://
    www.owl-ontologies.com/mason.owl#Semi-
    finished_part</crema:element>
            </crema:output>
          </crema:outputs>
          <crema:preconditions>
            <crema:element>( http://www.crema-
    project.eu/ExtensionA.owl#isLoaded  Ke1 Be1
    )</crema:element>
          </crema:preconditions>
          <crema:effects>
            <crema:expr type="and">
              <crema:element>( http://www.crema
                  -project.eu/ExtensionA.owl#
                  isNotLoaded Ke1 Be1)</crema:
                  element>
              <crema:element>( http://www.crema
                  -project.eu/ExtensionA.owl#
                  isLoaded Se1 Be1)</crema:
                  element>
            </crema:expr>
          </crema:effects>
        </crema:annotations>
      </bpmn:extensionElements>
      <bpmn:incoming>SequenceFlow_0mav5lz</bpmn
          :incoming>
      <bpmn:outgoing>SequenceFlow_1xn4g3t</bpmn
     :outgoing>
    </bpmn:serviceTask>
    ...
    <bpmn:sequenceFlow id="SequenceFlow_1kedmbi
        " sourceRef="ServiceTask_095jjui"
        targetRef="ExclusiveGateway_0vvoawp" />
    <bpmn:endEvent id="EndEvent_078m1ci">
      <bpmn:incoming>SequenceFlow_1l44fq6</bpmn
          :incoming>
    </bpmn:endEvent>
  </bpmn:process>
</bpmn:definitions>
```

Listing 2. Snipplet of BPMN v2.0 extension for IOPE semantic annotation of a task.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/
    MODEL" xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:camunda="http://camunda.org/schema/1.0/bpmn" xmlns:crema="
    http://crema.project.eu" xmlns:dc="http://www.omg.org/spec/DD
    /20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="
    Definitions_1" targetNamespace="http://bpmn.io/schema/bpmn">
  <bpmn:process id="Process_1" isExecutable="true">
    <bpmn:extensionElements>
      <crema:metadata>
        <crema:optimization>
          <crema:formulation>...</crema:formulation>
          <crema:results>...</crema:results>
        </crema:optimization>
        <crema:implementation>
          ...
          <crema:service implements="ServiceTask_033mvdo" seq="
              1">
            <crema:abstractService>
              <crema:marketplaceServiceID> 04917d1c-fd7a-4f36
    -9d87-e77c7106dfcb</crema:marketplaceServiceID>
            </crema:abstractService>
            <crema:concreteService origin="optimisation">
              <crema:marketplaceServiceID> f7817549-db5e-437c
                  -9077-370ce86302e5</crema:
                  marketplaceServiceID>
              <crema:owlsDescription> http://127.0.0.1:80/
                  oderu/Service/f7817549-db5e-437c-9077-370
                  ce86302e5.owl</crema:owlsDescription>
              <crema:bindings>
                <crema:binding>
                  <crema:origin>
                    <crema:env />
                  </crema:origin>
                  <crema:target>
                    <crema:variable name="Be1" service="
    f7817549-db5e-437c-9077-370ce86302e5" />
                  </crema:target>
                </crema:binding>
                <crema:binding>
                  <crema:origin>
                    <crema:variable name="Se1" service="
    308e5b62-a20e-4bfa-9931-7bd021ba3df8" />
                  </crema:origin>
                  <crema:target>
                    <crema:variable name="Se1" service="
    f7817549-db5e-437c-9077-370ce86302e5" />
                  </crema:target>
                </crema:binding>
              </crema:bindings>
            </crema:concreteService>
          </crema:service>
          ...
          <crema:service implements="ServiceTask_1ilmwjh" seq="
              1">
            <crema:abstractService>
              <crema:marketplaceServiceID />
            </crema:abstractService>
            <crema:concreteService origin="designer">
              <crema:marketplaceServiceID> f324fb92-9b1b-4af9
                  -a920-6078a66f2759</crema:
                  marketplaceServiceID>
              <crema:owlsDescription> http://127.0.0.1:80/
                  oderu/Service/f324fb92-9b1b-4af9-a920-6078
                  a66f2759.owl</crema:owlsDescription>
              <crema:bindings>
                <crema:binding>
                  <crema:origin>
                    <crema:env />
                  </crema:origin>
                  <crema:target>
                    <crema:variable name="Be1" service="
    f324fb92-9b1b-4af9-a920-6078a66f2759" />
                  </crema:target>
                </crema:binding>
              </crema:bindings>
            </crema:concreteService>
          </crema:service>
          ...
        </crema:implementation>
      </crema:metadata>
    </bpmn:extensionElements>
    ...
  </bpmn:process>
  ...
</bpmn:definitions>
```