Peer-reviewed version available at *Information* **2018**, *9*, 279; doi:10.3390/info9110279

Article

Smart Process Optimization and Adaptive Execution with Semantic Services in Cloud Manufacturing ‡

Luca Mazzola [1,*,†], Philipp Waibel [2], Patrick Kaphanke [3,†] and Matthias Klusch [4]

[1] School of Information Technology (Informatik), HSLU—Lucerne University of Applied Sciences, CH-6343 Rotkreuz, Switzerlandy; luca.mazzola@hslu.ch
[2] Distributed System Group, TU Wien, A-1040 Vienna, Austria; p.waibel@infosys.tuwien.ac.at
[3] EVANA AG, D-60325 Frankfurt am Main, Germany; p.kapahnke@evana.de
[4] DFKI—German Research Center for Artificial Intelligence, Saarland Informatics Campus D3.2, D-66123 Saarbrücken, Germany; matthias.klusch@dfki.de
* Correspondence: mazzola.luca@gmail.com or luca.mazzola@hslu.ch; Tel.: +41-41-757-68-90
† Luca Mazzola and Patrick Kapahnke worked at DFKI (German Research Center for Artificial Intelligence) during the ideation and development of the presented software solution.
‡ This manuscript is an extended version of the paper "***ODERU: Optimisation of Semantic Service-Based Processes in Manufacturing***" by Luca Mazzola, Patrick Kaphanke, and Matthias Klusch, Print ISBN: 978-3-319-69547-1, Online ISBN: 978-3-319-69548-8, doi:10.1007/978-3-319-69548-8_23, published in the proceedings of Knowledge Engineering and Semantic Web, Szczecin, Poland, 8–10 November 2017.

Abstract: A new requirement for the manufacturing companies in Industry 4.0 is to be flexible with respect to changes in demands, requiring them to react rapidly and efficiently on the production capacities. Together with the trend to use Service-Oriented Architectures (SOA), this requirement induces a need for agile collaboration among supply chain partners, but also between different divisions or branches of the same company. In order to address this collaboration challenge, we propose a novel pragmatic approach for the process analysis, implementation and execution. This is achieved through sets of semantic annotations of business process models encoded into BPMN 2.0 extensions. Building blocks for such manufacturing processes are the individual available services, which are also semantically annotated according to the Everything-as-a-Service (XaaS) principles and stored into a common marketplace. The optimization of such manufacturing processes combines pattern-based semantic composition of services with their non-functional aspects. This is achieved by means of Quality-of-Service (QoS)-based Constraint Optimization Problem (COP) solving, resulting in an automatic implementation of service-based manufacturing processes. The produced solution is mapped back to the BPMN 2.0 standard formalism by means of the introduced extension elements, fully detailing the enactable optimal process service plan produced. This approach allows enacting a process instance, using just-in-time service leasing, allocation of resources and dynamic replanning in the case of failures. This proposition provides the best compromise between external visibility, control and flexibility. In this way, it provides an optimal approach for business process models' implementation, with a full service-oriented taste, by implementing user-defined QoS metrics, just-in-time execution and basic dynamic repairing capabilities. This paper presents the described approach and the technical architecture and depicts one initial industrial application in the manufacturing domain of aluminum forging for bicycle hull body forming, where the advantages stemming from the main capabilities of this approach are sketched.

Keywords: Industry 4.0; XaaS; SemSOA; business process optimization; scalable cloud service deployment; process service plan just-in-time adaptation; BPMN partial fault tolerance

1. Introduction

As every other aspect of everyday life, also the manufacturing domain is strongly influenced by innovations in Information and Communication Technologies (ICT) [1,2]. Companies need to react flexibly to changing demands to remain competitive in a dynamic market [3]. The impact of ICT in this domain is broadly known as Industry 4.0 and ranges from the application of artificial intelligence in robot-assisted production to the usage of Internet of Things (IoT) devices, always connected and controllable just-in-time [4].

Traditionally, a Process Model (PM) is designed by the expert to represent in a standard language, such as the Business Process Model Notation (BPMN), an abstract representation of the *modus operandi* and the set of operations adoptable to achieve the expected goal. These elements basically translate into a set of flow objects to represent events (such as start, intermediate and stop), activities (practical elementary actions, called task) and gateways (conditions, or events, based on adaptation of the path, representing decision points). A particular instantiation of the PM, based on the relevant set of variables and conditions, takes the name of Process Instance (PI). To transform a PI into an enactable model, it is necessary to associate each task T with one or more services available to allow its execution in the physical world, achieving in this way its expected goal. Each service (or combination of them) used in this way is called the grounding service for the task T. The set of evaluated gateways and of provided grounding services for a full PI is known as the Process Service Plan (PSP). An additional requirement to support a full enactability of the PSP by an execution environment is the existence of a contextual environment to be used for service deployment, plus the presence of the variables' bindings amongst the set of grounding services, to support the exchange of all the information required for a correct service instantiation. During the process execution, meaning the ordered instantiation of the grounding services, a process registry, also known as a PSP log or execution log, is created by the RunTimeexecution environment and used to track the operations performed and their outcomes. Whenever one or more of the grounding services in a PSP become, temporarily or definitively, unavailable, we define this as a "broken" PSP. This situation requires a dynamic adaption to support the process execution completion, by providing an alternative set of available grounding services for the tasks that are lacking, using also the information provided by the PSP log.

There are multiple preconditions for allowing Industry 4.0 real-world applications; for instance, the need to define the domain formally in terms of ontological knowledge, the demand to give formalized representations of the executable services and the requirement of independence between business process models, their instantiation in the current context and the available services usable by the executed models. This calls for supporting tools that can provide an effective composition of services in the context of Everything-as-a-Service (XaaS) and Service-Oriented Architecture (SOA) systems, together with their Semantic variant, named SemSOA.

Along the same line, manufacturing business processes have to be designed and executed in a more dynamic production context, thus creating the need for adaptation and optimization at design time, as well as at runtime [5]. As a consequence, the design of process models for business applications has to go further than what the BPMN standard can support, as it needs to comprise representations for functional and non-functional requirements. This exceeds what can be specified in traditional Business Process Modeling (BPM) systems, which does not include semantic representations of product models and manufacturing services, as well as Key Performance Indicator (KPI) requirements and Quality-of-Service (QoS) aspects. Moreover, effective supporting tools need to be able to provide reliable model optimization to achieve the best executable PSP for business processes. Eventually, the provided PSPs should be designed to support a runtime incremental re-planning effectively, in case an included service is temporarily failing or becomes unavailable. Additionally, a sustainable approach requires just-in-time service leasing, their elastic deployment on request into the cloud, their monitoring and billing.

Due to the unavailability of solutions to tackle these issues in an integrated way [6], we developed a set of components whose cooperation can pragmatically solve the presented points. Starting from

the ontology necessary for the domain and business cases' description and reasoning, as well as the wrapping of services into their semantic characterization, the approach should be able to select the set of compliant services, available to implement the tasks. Subsequently, it should support the composition of functionally-correct PSPs based on semantic annotations, while optimizing their non-functional aspects formalized in terms of a Constrained Optimization Problem (COP). The resulting complete PSP is encoded back into specifically-developed BPMN 2.0 extensions. This approach partially bridges the gap between models and executable plans and provides at the same time the best variable assignments to optimize the outcome of the execution. Regarding the availability of the optimal PSP, a pragmatic tool for manufacturing in Industry 4.0 should provide an execution environment able to efficiently deploy the services grounded on the cloud, control them and react to eventual failure, with a smart re-planning policy.

The rest of the paper is organized as follows: In Section 2, the related work is presented, then Section 3 describes the set of components envisioned and developed in the CREMA framework; while Section 4 introduces an exemplary test case in the manufacturing domain. This includes a short overview of the scenario, followed by Section 5, with a brief description of the role of each of the components in this context. Then, Section 6 gives some initial thoughts about the modifications that were required to achieve our demonstrator and the extendability of the presented use case towards a pragmatic approach for Industry 4.0 in manufacturing. The conclusions are eventually given in Section 7.

2. Related Work

Multiple different domains are affected by our proposed approach. This section gives a brief overview of their current status, in particular with respect to the following themes: SOA and its semantic variant used for service matching and composition; the XaaS approach; business process optimization by user-defined KPI and QoS metrics; PSP composition, including variable bindings and optimal configuration; elastic process execution in the cloud; aspects of fault tolerance in business processes' realization; and deployment of container-based software as a supporting solution for heterogeneous services' enactment.

A semantic service (also known as semantic web service) is an approach to support the automatic interpretation of the service functions for service-based systems or intelligent agents [7]. Its central idea is to use standardized annotations to conceal the functional and non-functional semantics from software agents defined for different tasks, not only in a machine-readable way, but also machine-interpretable. In order to achieve this interoperability between heterogeneous components and to allow them to consider any semantic service they can encounter, the semantics should be defined using concepts and rules coming from a shared ontology. The ontology itself needs to be formally defined adopting a widespread format, such as one of the W3C standard languages, OWL2 or RDFS. Applications and agents are consequently in the position to be able to rely on these well-founded formal semantic annotations for their service interpretation, with the final aim of discovering required services with the indicated high-precision or being able to plan a complex task by composing elementary services in an automated way. There are a number of currently notable frameworks for semantic service description, each one of them with some advantages and some limitations. Examples of them are OWL-S, WSML, the W3C standard SA-WSDL and USDL.

Another important paradigm at the foundation of the present work is the so-called everything-as-a-service [8]. It represents an abstraction layer over the actual resources, wrapping them into well-defined public interfaces to support every possible operation, such as search, selection and invocation. On top of this, a XaaS approach clearly separates the service concrete instantiation from its semantic published description, guaranteeing a complete separation of the model and the service-based plan implementing it. For these reasons, this paradigm is widely adopted in the field of cloud computing infrastructure.

The archetype of the semantic service-oriented architecture, by composing the semantic aspects and the XaaS, is applicable also to the manufacturing domain, allowing specialized process models' consideration. In this way, it is possible to adopt a proper procedure for semantic service discovery, selection and composition planning, resulting in the creation of a fully-automatic implementation based on the available semantic services.

The key idea is to enable automated understanding of task requirements and services by providing semantic descriptions in a standardized machine-understandable way by using formal ontological definitions [7], for example in OWL2 (W3C standard: https://www.w3.org/TR/owl2-overview/). In [9], the authors proposed SBPM, a framework to combine Semantic web services and BPM to overcome the problem of automated understanding of processes by machines in a dynamic business environment. Similarly, the authors of [10] proposed sBPMN, which integrates semantic technologies and BPMN to overcome the obvious gap between an abstract representation of process models and actual executable descriptions in BPEL. The work in [11] follows the same track with the proposal of BPMO, an ontology, which is partly based on sBPMN, while [12] took sBPMN as the basis for the Maestro tool, which implements the realization of semantically-annotated business tasks with concrete services by means of automatic discovery and composition. In [13], a reference architecture for semSOA in BPM was proposed, which aims to address the representation discrepancy of business expertise and IT knowledge by making use of semantic web technologies. All of these proposals rely on formalization different from (although based on) BPMN or do not aim for a full integration from a formalism point of view. In the work [14], the authors proposed an approach that uses BPMN extensions to add semantic annotations for automatic composition of process service plans and to verify their soundness, but this approach does not consider QoS-aware or runtime optimization. Adopting a similar approach, our demonstrator proposes a set of BPMN extensions that not only enable interoperability by offering process model composition, task service selection and process execution, but also provides a way to represent the best values to optimize the QoS and the quality values achieved.

Our optimized PSP creation component applies state of the art semantic service selection technologies [15] to implement annotated process tasks. Non-functional criteria, often referred to as QoS (e.g., costs, execution time, availability), can additionally be considered to find matching services in terms of functional and non-functional requirements [16,17]. Here, optimality with respect to the non-functional QoS specifications is achieved at the process model level by solving (non)linear multi-objective COP (muCOP) as an integrated follow-up to the pattern-based composition.

Most existing approaches to PSP composition do not cover the combination of functional (semantic) aspects and non-functional (QoS-aware) optimization. For example, [12,18**?**] considered functional semantic annotations to implement business processes by means of a service composition plan.

The work in [19] provided a survey giving an overview of existing approaches and initiatives in this direction and highlighted research questions. Integrated functional and non-functional optimizations have rarely been considered, with the notable exception of [20]. While composition typically includes the computation of possible data flows, our proposed approach additionally finds optimal service variable assignments that are also required for executing the resulting plans. This is a feature not yet considered by existing work. Moreover, our PSP computation component is equipped to perform re-optimization of PSPs at runtime upon request, which is also a novel feature. Finally, our optimization component employs the means of RDF stream processing to react to service changes (non-functional QoS aspects) reported by the service registry. This information can be used to trigger optimizations pro-actively if the RDF stream engine identifies that a previously computed PSP is affected.

Another area of recent innovation is the one of micro-services, which stems from the very diffused service abstraction. Here, many innovations arose in the last few years, such as all the family of techniques for container-based deployment [21]: Docker (https://www.docker.com), rkt (short for *CoreOS's Rocket*: https://coreos.com/rkt/) or LXC (Linux Containers: https://linuxcontainers.org/).

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

They represents virtual machines (VM), but in contrast with the historical ones, they are lightweight, devoted to bundling together every requirement for a service, allowing a zero-configuration on deployment. This means all the software and operating system dependencies and all the service configurations are already contained and pre-built. Despite the original scope of lightweight VMs for the micro-service domain [22], not much time passed before their usefulness was appreciated in other domains, such as legacy services' integration [23]. In our scenario, Docker is used to facilitate the integration of heterogeneous services into business processes. An additional feature that makes containers a natural choice in cases where the execution is on-demand is the better startup performance coupled with a lower resource footprint, in comparison to established virtual machines [24].

Not too much research was devoted till now to exploring the theme of elastic process execution [25]: here are the most relevant publications, in this respect. ViePEP (Vienna Platform for Elastic Processes) is an eBPMS (elastic Business Process Management System) created to fuse traditional process engine functionality with a cloud controller [26,27]. This means that relying on cloud resources, this solution allows executing software-based processes and instantiating process tasks on them. Additionally, ViePEP offers the possibility to optimize the service enactment by using in the most cost-effective way the readily obtainable cloud resources, respecting the predefined Service Level Agreements (SLAs) [28]. ViePEP is based, in common with our approach, on software utilities to represent the execution elements of a process task, but does not offer any feature for the automatic selection and composition of the existing services. Consequently, our solution emphasizes the capabilities for matching and composition of automatic service and, in a failure case, of process optimization at runtime.

Juhnke et al. [29] provided another similar work: to provide process tasks' enactment, they used on-demand VM cloud-based computational resources. Anyway, they relied instead on a BPEL-based process representation. Our solution is instead based on BPMN v2.0 extensions, and this supports the interpretation simplification of the executable process service plans.

Other works that adopt VMs to implement the process tasks composing business processes on cloud resources were Wei and Blake [30], Bessai et al. [31] and Cai et al. [32]. However, each of them lacked automatic service selection capabilities. This aspect could generate an issue during the process execution, as in the case of a necessary reconfiguration for service unavailability, it lacks the minimum level of flexibility necessary to support runtime dynamic optimization, in an automated fashion.

Another relevant area for the current work is the so-called "cloud manufacturing". An example of work in this areas is from Chen et al. [33], where they introduced a novel cloud manufacturing framework with an auto-scaling capability. The main differences with our work start with their approach of transforming single-user manufacturing functions into multiple contemporary usage cloud services, where we rely on semSOA to abstract from the underlying rooting function into a semantic rich service. Secondly, in the work of Chen, the optimality was obtained in terms of the minimal number of VMs required to confine the average service time to lower than a predefined threshold. This is much more inflexible than our approach, where the user can define any objective function for the optimization problem. Obviously, this flexibility comes at a cost, in this case the need for the user to understand clearly and coherently formulate the COP and the computation time required by our solution to provide an optimal solution.

3. CREMA: Towards Industry 4.0 for Manufacturing

The objective of this work is to provide a pragmatic solution for implementing an Industry 4.0 approach in the manufacturing domain. This is based on the smart process composition supported by the usage of semantic services, together with the possibility to optimize the service plan through a novel definition of requirements and objectives. That is facilitated by an ad-hoc developed COP language for defining QoS-based functions, which can be embedded into BPMN extensions. As a dynamic and just-in-time adaptation to the frequently-changing execution context and service availability, the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 November 2018 doi:10.20944/preprints201810.0143.v2

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

proposed approach provides an adaptive instances execution, by automatically dealing with "broken" PSPs and repairing them seamlessly using services, or a combination of them.

In the following sections, we present each individual component required to implement this vision: we start by the ontology and its usages (Section 3.1), followed by a short depiction of the helper UI for the semantic services annotation (Section 3.2). With these two elements in place, it is possible to define the Process Model (PM) in BPMN, together with the additional elements that define its semantic meaning, by the usage of an extended BPMN editor (Section 3.3). When the user requires the execution of the process, an optimized PSP is computed by the system, through semantic service matching and COP solving on the QoS-defined objective function (Section 3.4). This PSP is then executed by the runtime environment (Section 3.5). The runtime environment then uses the invocation and controlling capabilities of the deployment component (Section 3.6), which controls the retrieval, instantiation and feedback collection of the services on the cloud resources.

3.1. Ontology Exploration, Validation, Extension and Editing

For our solution described in this paper, we propose a reference domain ontology called "CREMA Data Model, Core module" (in short, CDM-Core) [34], which provides OWL2 descriptions of concepts from the manufacturing domain. The released version is publicly available under the Creative Commons license (CC BY-SA 3.0) at https://sourceforge.net/projects/cdm-core/ and concentrates on hydraulic metal press maintenance and car exhaust production, but for the current work, we designed an extension to cover the aluminum forging-based injection process. As an example, Figure 1 reports the OWL representation of the concept "Robot" using an abstract syntax, for readability reasons.

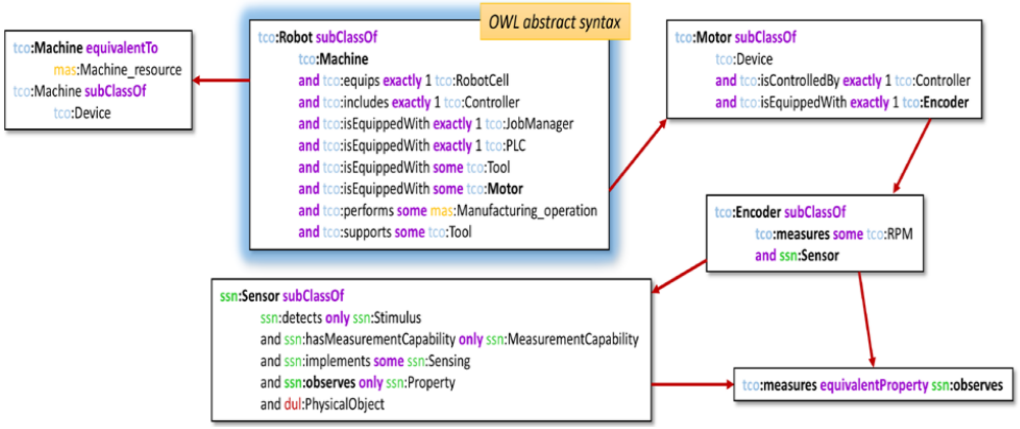

Figure 1. The definition of the concept "Robot" in the developed ontology, represented using the OWL abstract syntax. Here, the expressive richness of an ontology is evident, by allowing the expression of complex knowledge. A *Robot* equips exactly a *RobotCell* and includes a *Controller*, plus being completed by some *Tool* (of the supported types) and *Motor*. It performs some *Manufacturing_operation*.

Additionally, the ontology can be used for creating an enriched version of the industrial data stream from IoT devices, such as in the case of Figure 2, where a multi-reader device provides a set of values for a controlled pneumatic circuit. In this particular example, it is possible to see how the raw (tab separated) input is transformed into an RDF (linked data enabled) fact. Here, the flow concentrates only on the field marked in red, to show how every single measurement existing in the stream is transformed. As modifying an ontology is not a simple task, in particular, due to the strict formalism required, we also developed a very lightweight helper interface [35], to provide minimal support in this task to domain experts and business-oriented process modelers. This is a very initial effort towards better sustainability and acceptance of ontologies and formalized domain knowledge as a base for industrial and business modeling efforts. Figure 3 presents a small extract of its representation

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 November 2018 doi:10.20944/preprints201810.0143.v2

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

capabilities, showcasing the translation of a CDM-Core segment in RDF/XML syntax (in the small overlapping window) into a set of graphical objects.

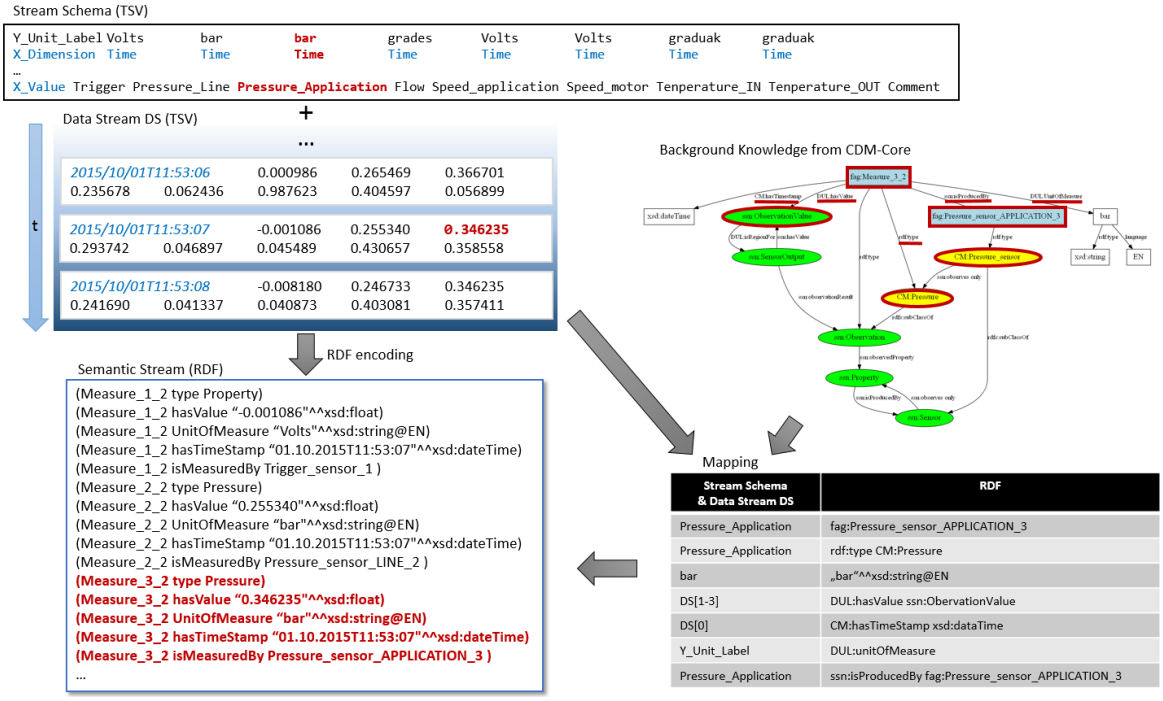

Figure 2. The use of the CDM-Core ontology for a data stream semantic enrichment (RDFication).

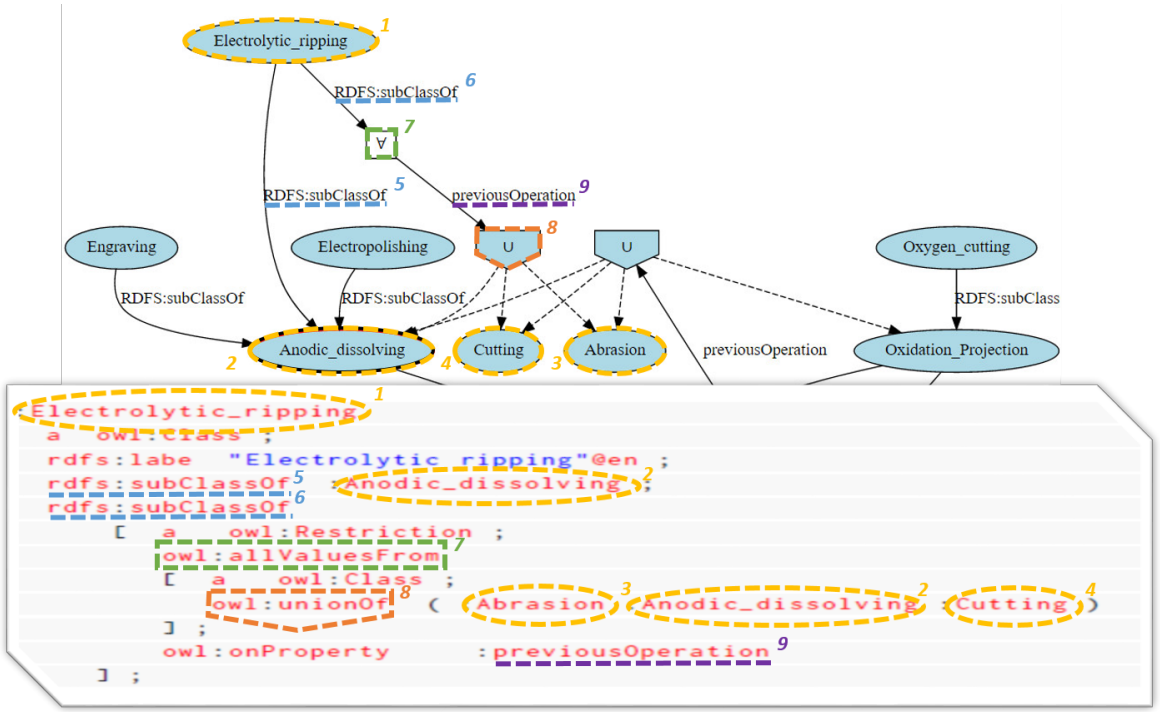

Figure 3. Extract of the lightweight web interface for ontology exploration and manipulation. In the superimposed window, the underlying ontology extract (in RDF/XML syntax) is presented together with the visual object correspondences, highlighted by colors (function) and numbers.

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

3.2. Service and Task Annotation

In order to enable a XaaS abstraction, all services require being wrapped with semantic annotations of their external behaviors. For this, the W3C recommendation OWL-S [36] is used, which provides a means for not only Inputs, Outputs, Preconditions and Effects (IOPE) annotations, but also for the QoS aspect required by the non-functional optimization. QoS aspects are not predefined in OWL-S, but can be adapted flexibly to the specific use case at hand. Definitions for various QoS aspects are defined in the CDM-Core ontology (or can be defined based on it in terms of extensions) and could, for example, represent monetary costs of using a service, operation cycle time of a machine or accumulated failure probability.

Figure 4 presents the basic UI provided by our architecture for working with service semantic annotation. In the bottom section, the IOPE annotations are visible, together with the variables' section, which allows binding environmental variables to the service execution, for runtime usage. The over-imposed boxes give an impression of a potential instantiation for each semantic annotation element. Additionally, the round overlay shows the search interface, to support the search and management of available semantic concepts (and relationship) names from the CDM-Core ontology. At the moment, no semantic checking of compatibility is performed, as this will heavily overload the solution.

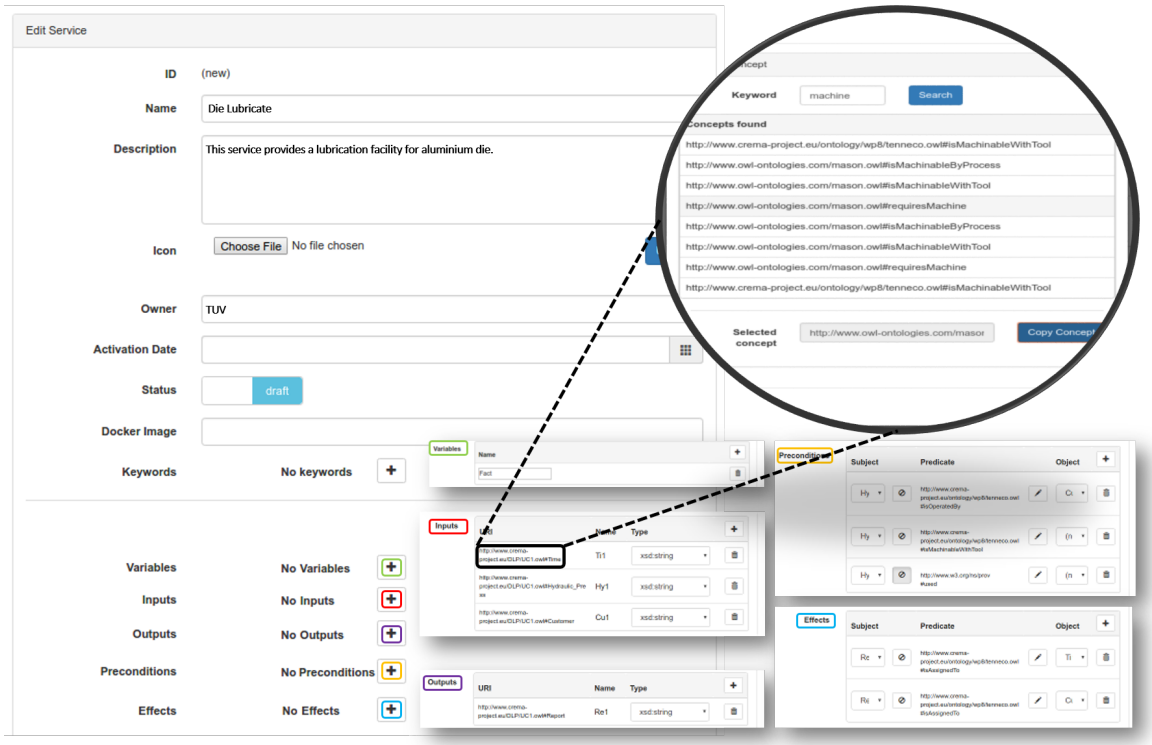

Figure 4. A screenshot of the service and task annotation tool, with some example sections for the variables and IOPE annotations. In the round over layer, an example of the search for concepts and the relationship in the semantic space provided by the CDM-Core ontology.

Peer-reviewed version available at *Information* **2018**, *9*, 279; doi:10.3390/info9110279

3.3. Process Model Composition, Annotation and Parametrization

Once the semantic source for the domain definition has been specified and the semantically-annotated services have been created, the process models can be created. In order to be able to compose automatically functionally valid process service plans given a process model, it is necessary that process tasks be equipped with structured semantic descriptions. Following the SemSOA approach, the IOPE of tasks are described in terms of formalized ontological domain knowledge.

These are basically BPMN models, enriched by a set of annotations to define the semantic behaviors of each task, in terms of IOPE. This means that the semantic annotations are embedded in the BPMN model by making use of extension elements at the task level. At this stage, a default semantic service can optionally be bound by the process designer, when considered useful as a default and zero-effort option for the process implementation. Additionally, the editor allows adding variables that can be used during the process instantiation and exploring the produced XML encoding for the process model, for debugging purposes with respect to BPMN. Figure 5 presents a screenshot for the CREMA process model editor, with a simple model for forging the single hull of an aluminum-based bike frame by injection.

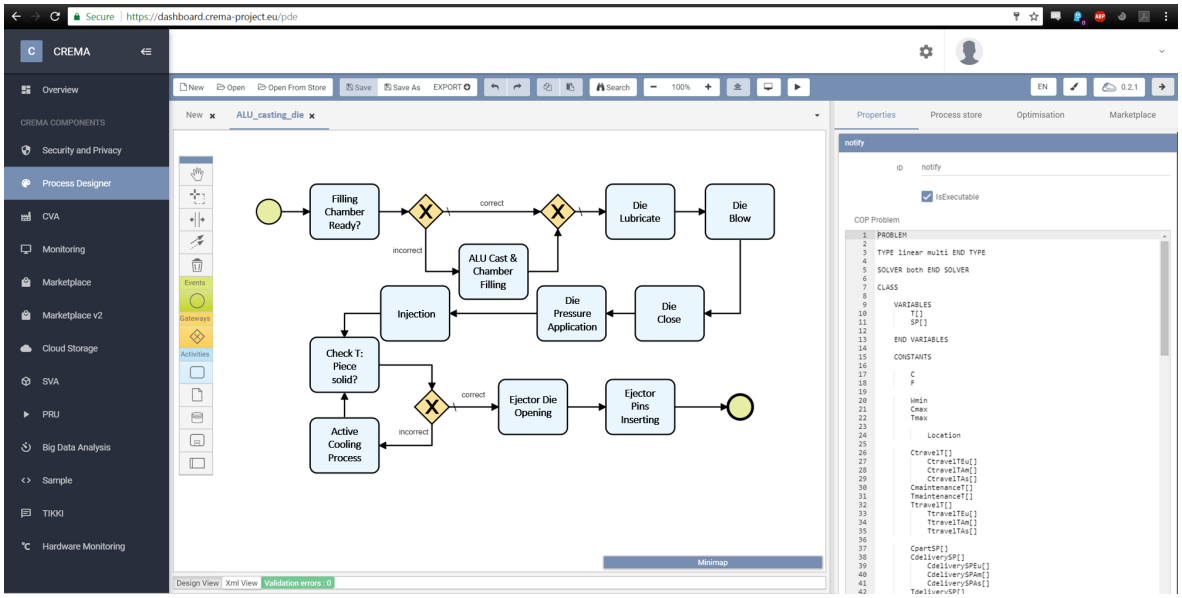

Figure 5. A screenshot of the enriched BPMN editor: on the right side, a designed process model, and on the left (partially visible), the definition of the associated constrained optimization problem.

3.4. Process Optimization by COP Solving with Semantic Services

To provide a service-based solution, we developed a one-stop process service plan composition and optimization component for extended BPMN [37]. In order for the proposal to be optimal with respect to the set of possible functionally valid solutions, it has to make particular choices driven by non-functional requirements, which are expressed as functions of the QoS measures provided by the services. Moreover, it computes concrete settings of service input parameter values, which yield optimal results in terms of the optimization criteria.

This is done by specifying a COP at the process model level, whose solutions dictates what services to choose from and what parameter settings to use when calling services. The COP formulation includes information on how to map optimal parameter values to service inputs and service QoS to COP constants. The outputs produced by the optimization component are PSPs encoded in the original BPMN itself by making use of BPMN extensions. Besides the optimal services and input values for calling the services as described above, this also includes possible data flows with parameter bindings

among services. Such a PSP implementing the process model can then be instantiated at runtime by a process service plan execution environment.

To achieve this objective, the optimization component follows two steps in a sequential manner: (a) it performs a pattern-based composition using semantic service selection for all semantically-annotated process tasks and the computation of possible data flows. Then, it executes a (b) QoS-aware non-functional optimization by means of COP solving at the process model level. This second step selects particular services out of sets of functionally-fitting services per tasks previously identified and provides the optimal settings for service inputs.

This workflow can be applied at design time and runtime (of a process model execution instance). At design time, the optimization component will be called after a process model has been defined in order to provide an executable implementation of the model as guidance for the execution environment. The runtime case appears as soon as a PSP is executed. Additionally, the execution environment can query back the optimization to provide alternative PSPs in the case of an exception during execution (e.g., a service becoming unavailable or failing). For this, the plan-enacting tool should not only provide to the optimization component the PSP it tried to execute, but also the current state of execution. This includes information on what services have already been executed, how gateways have been evaluated and what services caused errors during execution.

The aim of this component in the runtime case is then to provide an alternative solution for the given process instance. That is, it tries to patch the existing PSP and considers the current state of the world as fixed and not undoable, nevertheless trying to re-implement in an optimal way the part of the process model still uncovered or not correctly executed.

3.4.1. Constraint Optimization Problem Definition

We defined a context-free grammar COPSE [2] to represent constrained optimization problems by use of antlr4 (http://www.antlr.org/) (cf. Listing 1). The COP specification starts with the definition of its type (linear vs. non-linear, single vs. multi-objective, etc.) and continues with the declaration of the problem class.

In this part, the variables, constants and functions are indicated, while in the last segment, any complex function can be defined using operators such as *MAX*, *MIN*, *SUM*, *PRODUCT* and *IF-ELSE*. The set of constraints is then defined with respect to the variables, constants and functions already specified, and the objective function(s) is normally constructed by *minimizing* one or more functions (or functions combination). In the case of a multi-objective, it is possible to have many of them, also in a combined form of a *MIN-MAX* COP problem.

The COPSE^2 grammar also allows mapping back the achieved value to the produced PSP into a semantic concept. In the second part of the constraint optimization problem definition, the current problem instance is indicated: after defining the variables' domain and the value of the constants, the mapping of variables' values that give the optimal solution is reported back to semantic concepts used as the inputs of the services used.

This approach allows the definition of complex aggregates of QoS and environment variables instead of mere lists of objectives for simple QoS, extending the expressive capability with respect to the non-functional optimization problem definition. In [38], we showcased how this flexibility can be useful to represent heterogeneous optimization problems.

Peer-reviewed version available at *Information* **2018**, *9*, 279; doi:10.3390/info9110279

Listing 1. antlr4 grammar for Constraint Optimization Problems.

```
grammar COPSE2_meta;

problem: 'PROBLEM' type solver problemclass probleminstance output? 'END PROBLEM';

type: 'TYPE' Linear Objective 'END TYPE';
Linear: ('linear'|'nonlinear');
Objective: ('single'|'multi');

solver: 'SOLVER' Solver 'END SOLVER';
Solver: ('centralized'|'distributed'|'both');

problemclass: 'CLASS' variables constants? functions? constraints? objectivefunction+ 'END CLASS';

variables: 'VARIABLES' (Identifier|ArrayIdentifier)+ 'END VARIABLES';
constants: 'CONSTANTS' (Identifier|ArrayIdentifier)+ 'END CONSTANTS';
functions: 'FUNCTIONS' function+ 'END FUNCTIONS';
functionSign: Ident '(' identList ')';
function: functionSign '=' (expr|ifexpr);

Comparison: '>='|'<='|'=='|'!='|'>'|'<';
Assignment: '=';
expr: '-'? term (('+'|'-') term)*;
term: mterm (('*'|'/'|'^') mterm)*;
dim: Ident'.length' ;

loop: ('SUM'|'PRODUCT') '(' Ident ',' (Number|dim) ',' (Number|dim) ',' expr ')';

mterm: (Ident|ArrayElem|REAL|'(' expr ')'|('MIN'|'MAX') '{' expr (',' expr)* '}'|functionSign|dim|Number|loop);

ifexpr: 'IF' expr Comparison (expr|Number) 'THEN' (expr|ifexpr) 'ELSE' (expr|ifexpr) 'END IF';

constraints: 'CONSTRAINTS' constraint+ 'END CONSTRAINTS';
constraint: expr (Comparison|Assignment) (expr|Identifier|Number);
```

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279


```
34
35 objectivefunction: ('minimize'|'maximize') expr ('->' URI)?;
36
37 probleminstance: 'INSTANCE' variabledomains? constantvalues? 'END INSTANCE';
38
39 variabledomains: 'DOMAINS' vdomain+ 'END DOMAINS';
40 constantvalues: 'VALUES' cvalue+ input? 'END VALUES';
41
42 input: 'INPUT' inputEntry+ 'END INPUT';
43 inputEntry: Identifier '<-' '(' Identifier ',' URI ')';
44 URI: 'http://' ([a-zA-Z0-9/.])+ '#' ([a-zA-Z0-9])+;
45
46 vdomain: (Ident|ArrayIdent) ( Number | '[' Number ',' Number ']' | '{' Number (',' Number)* '}');
47 cvalue: (Ident|ArrayElem) Assignment Number;
48
49 output: 'OUTPUT' (valueAssignment|serviceSelection)+ 'END OUTPUT';
50 valueAssignment: (Ident|ArrayElem) '->' '(' Identifier ',' URI ')';
51 serviceSelection: ArrayIdent '::' Ident;
52
53 fragment Letter: [a-zA-Z];
54 fragment ANumber: [0-9];
55 fragment INF: ('INF'|'-INF');
56
57 Number: (('-'? (ANumber+|ANumber* '.' ANumber+) ('*' ('10'|'e') '^' '-'? ANumber+)?)|INF);
58
59 Ident: Letter (Letter|ANumber|'_')*;
60 ArrayIdent: Ident'[]';
61 ArrayElem: Ident'['Ident']';
62 identList: Ident (',' Ident)*;
63
64 WS: [ \t\r\n]+ -> skip;
```

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

3.4.2. Process Service Plan

The computation of a PSP is presented in Algorithm 1, which uses four helper functions. The first one is **SIM** ($IOPE_A$, $IOPE_B$) in Line 10, which is used to compute the similarity between two IOPE annotations based on a selected measure. Given the semantic description of a task ($IOPE_A$) and a service ($IOPE_B$) as the input, the adopted measures consider a logic-based signature plugin match for inputs and outputs and a logic specification plugin for precondition and effects. These matching filters are inspired by the classical plugin matching of components in software engineering. While a plugin match is commonly considered near-optimal, we prioritize services with semantic descriptions, which are logically equivalent with respect to the requested functionality. A possible ranking of logic-based semantic matching filters is proposed for iSeM, as shown in [39]. Alternative approaches to semantic service selection learn the optimal weighted aggregation of different types of non-logic-based and logic-based semantic matching filters [40].

Algorithm 1: The pseudocode for the process service plan composition.

Input: PM: semantically annotated BPMN model, **S**: set of available services
parameter: Sim$_{min}$: minimal similarity value accepted
Output: PSP: the computed process service plan

1 **forall** $s \in S$ **do**
2 $\quad IOPE_s \rightarrow IOPE_S$;
3 **end**
4 **forall** $task \in PM$ **do**
5 $\quad task \rightarrow T$;
6 **end**
7 % Find task service candidates
8 **forall** $t \in T$ **do**
9 $\quad$ **forall** $s \in S$ **do**
10 $\quad\quad$ **if** $SIM(IOPE_t, IOPE_s) >=$ **Sim$_{min}$ then**
11 $\quad\quad\quad s \rightarrow CANDIDATES_t$;
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 **end**
15 % Solve the COP
16 **forall** $t \in T$ **do**
17 $\quad$ **forall** $s \in CANDIDATES_t$ **do**
18 $\quad\quad$ **forall** $QoS \in T$ **do**
19 $\quad\quad\quad QoS \rightarrow Parameters_{s_t}$;
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 **end**
23 $Solutions =$ COPSOLVE(Parameters);
24 % For all the Pareto-optimal solutions, compute a valid data flow
25 **forall** $Solution \in Solutions$ **do**
26 $\quad$ COMPOSEVARIABLEBINDINGS(Solution) $\rightarrow Plans$;
27 **end**
28 % Return a process service plan using the first solution
29 **PSP**=MERGEPMWITHSOLUTION(PM, $Plans$[0]);
30 **return PSP**;

A second helper function is the **COPsolve** (parameters) used in Line 23 for computing the set of Pareto-optimal solutions of the COP. This is a simple compiler that transforms our COP definition into a running instance of a JaCoPsolver (http://jacop.osolpro.com/), using the set of parameters given.

The call to **ComposeVariableBindings** (solution) computes a possible set of variable bindings, which together define the data flow (Line 26). Bindings are determined by checking the semantic compatibility of the semantic variable types. This ensures a functionally-meaningful assignment beyond simple data type compatibility checking. The overall aim of this function is to connect as many service inputs in the solution with the outputs of services earlier in the execution order determined by the process model definition. Inputs that cannot be bound in that way are considered environmental variables. This ensures the direct executability of the computed service plan.

Please note that the pseudocode leaves out details on the handling of gateways and different possible execution paths through the process model for parallel execution and choices. Without loss of generality, the different paths can be considered additional options for generating PSPs, each indicating other gateway decisions and a valid data flow given this decision. The component is able to handle parallel (AND), choice (OR) and exclusive (XOR) gateways. While the AND gateway opens up independent parallel paths and is easy to handle, the XOR and OR gateways result in n and $n!$ possible alternative execution paths, thus widening the problem space significantly. Structurally, however, all these options are handled in an analogous way to what was explained.

Eventually, **MergePMwithSolution** (PM,Plan) takes care of adding the full metadata section into the original process model to create an executable PSP. This happens at Line 29.

Functional optimization (services' selection): The first step for creating a PSP is to select all the possible candidates functionally valid for each task. We rely on functionally equivalent *exact* or on *plugin* matches [41] that are limited to direct subclass relationships. This way, all PSPs' logical properties (in terms of IOPE) are conserved with respect to the given PM.

This step creates for each task a set of candidates, either a simple or composed service. In fact, the selection of their best composition is left for the non-functional optimization, based on the COP solution. Only after this additional phase, the actual service implementation in the returned PSP is complete.

Non-functional optimization (optimal services composition): Amongst all the possible combinations of services of the candidate pools of the process tasks, the best (or Pareto-optimal in the case of a multi-objective problem) option is chosen as part of the overall solution. This implies solving the COP problem associated with the PM, such as the example in Listing 2, by minimizing the function **TotalCost(X)**. For an introduction to the BPMN extensions defined in CREMA and used by our components, we refer the reader to [42].

3.4.3. Optimality of the Produced Solutions

The produced service plan, together with its variables' bindings and contextual optimal settings, is consequently optimal with respect to both the functional and non-functional aspects of the process model. In fact, it implements each task present in the model with a single, or a composition of, semantic service, which are equivalent or compatible (plugin) with the task annotations. This guarantees its functional optimality at the global level, as all the semantic description are respected, fulfilling in this way the global IOPE footprint and the internal variable chaining requirements. Anyway, at this level, is possible to have completely equivalent plans, for example implementing the same tasks with alternative equivalent services, but with different QoS parameters.

At the non-functional layer, instead, the optimality is translated into two planes: initially, the best performing service amongst the potential alternatives for each task to be really executed (considering also the already performed services and the gateway branches, in case of runtime replanning) is discovered based on the COP formulation. Secondly, using the set of services identified, the bindings of produced outputs and required inputs in subsequent services are semantically determined, and the assignment of the best parameters for contextual variables in the execution environment is

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

decided. This produces a well-defined, non-dominated, value for the optimization function, which is communicated to the user, for reference and validation.

3.4.4. The Optimization Facility in CREMA

Based on the SOA approach, the optimization facility is a JAVA-based software implemented as a RESTful service. Figure 6 depicts its basic components and the interactions it requires for a fully-functional process enterprise execution platform, such as the one that the CREMA solution provides.

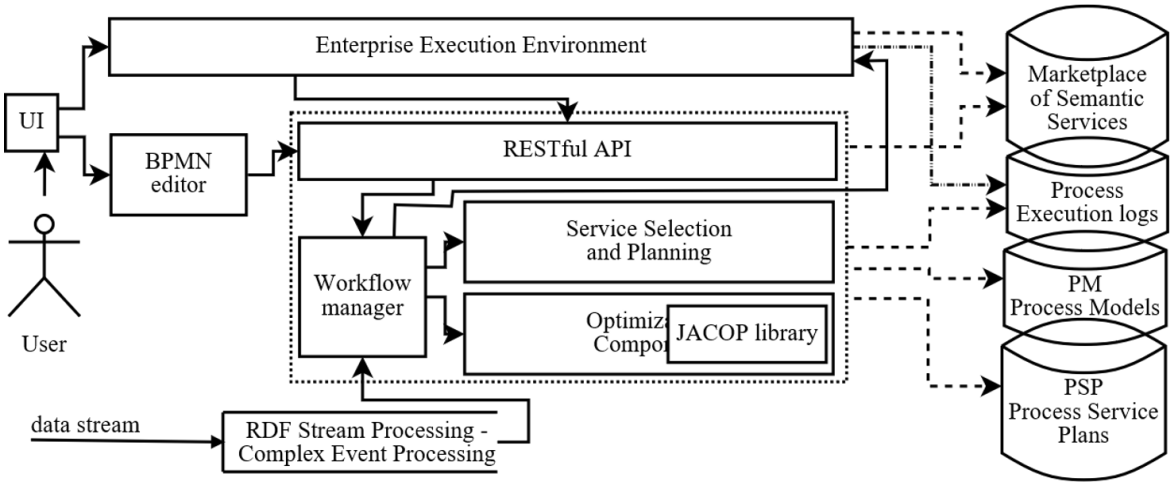

Figure 6. The optimization component in the context of a fully-fledged BPM and execution architecture.

To achieve this objective, during the CREMA project, a set of functions was designed and implemented: they allow asking for an integrated composition and optimization (meaning, considering both the functional and non-functional requirements specified in the input BPMN) or separately, in the case when (a) the user is interested only in a functionally valid plan or (b) when a composed plan already exists that requires being optimized based on the non-functional QoS measures and the user-defined objective function(s). This is valid both at design time (input is a process model) and at runtime (input is an instance of the process model, together with the execution log, if available). For accountability, then, a function to allow the user approval of the computed PSP is provided. Additionally, it exposes also a set of utility operations, ranging from operations able to retrieve the ordered list of services found to implement a single task, till capacity for fetching previously computed PSPs, to support user inspection for alternatives, if of interest.

3.5. Process Service Plan Execution and Contextual Environment Management

Our demonstrator also envisioned a component responsible for the execution of a PSP. It is based on an affirmed process engine, and its role is to execute process tasks according to the order defined in the process. As our approach is based on an optimal PSP realization by the optimization component, the execution follows precisely the service sequence defined in the process service plan. This is our instantiation of the flexible service selection concept for process tasks' execution.

This additional facility is bounded to the extension of parsing abilities and deployment capabilities for a standard process engine. This arises because the communication of the suggested service sequence and the externalization of the optimal environment initialization required for its implementation are performed by the optimization component using BPMN extensions; and our approach needs consequently to implement the logic necessary for considering this additional PSP section at the process engine level.

Figure 7 shows its main UI, with possible plans to be executed, their creation time (for reference) and the current number of instances running. Furthermore, from this interface, the human operator can retrieve information and control the process, by launching, stopping, pausing or resuming instances of it.

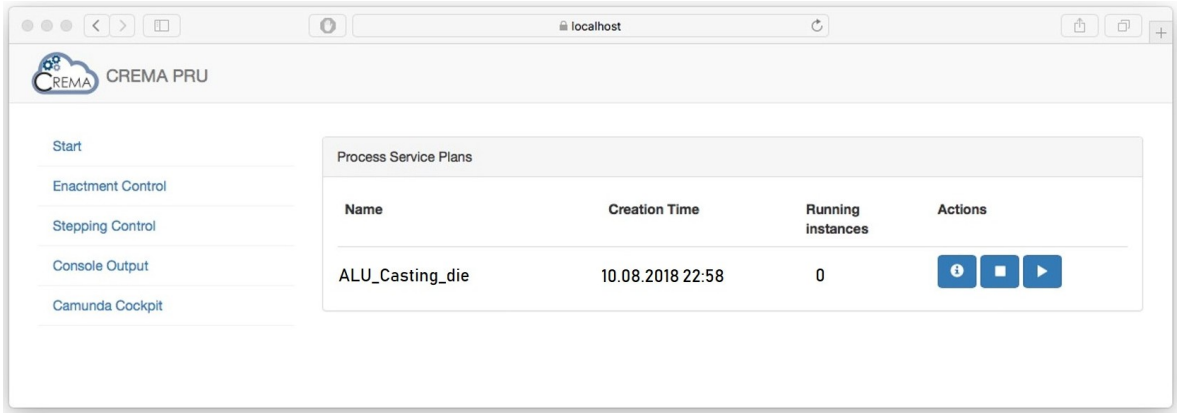

Figure 7. Screenshot of the process execution engine showing the process "ALUCasting die", whose instance can be executed.

In case of issues with the process execution, this component is additionally able to capture this exception automatically, to pause the execution, to invoke the interaction with the optimization component, to request a new PSP respecting this additional previously unforeseeable constraint and, then, to continue the execution of the paused process with the updated PSP. For example, such an exception can happen during the execution of a service, such as in the cases of a hardware breakdown of a manufacturing machine, a physical resource (e.g., a human operator) already busy or the unavailability of a service for unplanned maintenance reasons. The possibility of providing this just-in-time adaptation of the PSP offers an initial fault-tolerance capacity for process execution, which is a novelty of our approach.

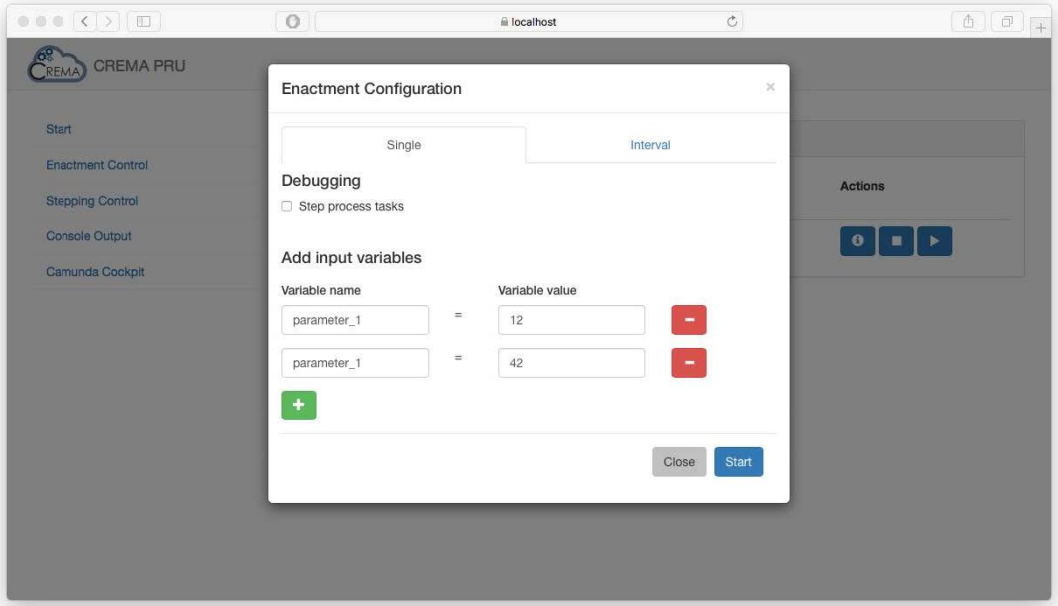

Figure 8. Screenshot of the process execution engine "enactment configuration" capabilities: here, the user can launch one time or repeatedly on an interval basis a process, passing an arbitrary number of contextual variables, to be used as the complement for the process enactment environment.

For debugging capabilities and for testing purposes, the component can also enact (once or repeatedly on an interval base) a process using a manually-defined configuration, as shown in Figure 8. This supports the human operator in discovering the reason for unexpected outputs or the behaviors of the process executed.

3.6. Process Deployment in the Cloud and Execution Control

At the lower level of the control chain for process execution sits a service deployment and execution control facility. This is responsible for the on-demand enactment of services on cloud resources. Differently from purely informative business PMs, the manufacturing domain is characterized by a natural mix of heterogeneous kinds of services. Naturally, there are traditional software services, such as analysis of values or computation of control conditions, that are naturally enactable on computational resources.

Typical of this domain, there are also real-world services, such as the one that physically manipulates the material and semi-finished piece to produce the final outcome of the manufacturing activity. These can be welding machines, robot arms, manipulators, numerically-controlled machines (CNC), lubrication systems, and so on. It is necessary to transform them by adding a software-based representation that works as their digital interface to enable their deployment and control on cloud resources.

Human-based services are another typical category and can be, for instance, the task of loading or unloading parts, manipulating the machines, managing an unexpected condition or collecting information about an environmental condition that is not automatically monitored. For this type of service to be enactable in a distributed computation environment, it is necessary to design some user interfaces that act as a human communication vector. This additional interfacing facility allows, on the one side, providing instruction and input and, on the other end, collecting feedback and information for reporting back to the execution context.

In order to combine these services, we have designed an abstraction approach called Proxy Service Wrappers (PSW), which provide a uniform representation of every different kind of service, allowing their usage as a grounding for process tasks.

The basic foundation of a PSW is a set of requirements that need to be implemented by services in order to be integrated. Firstly, there is the need to expose two different endpoints for each service: an endpoint for checking the *Availability* of the actual service, which should indicate its current leasability, taking into account all the limitations affecting it (e.g., contemporary usages, stale states or maintenance operations), and a *Start* one, which deploys the service using a JSON-encoded input parameter object. As a precondition, it is possible to call the latter one only under a positive answer from the former endpoint invocation. As a consequence of a *Start* endpoint triggering, a PSW starts the operation for a software-based service, triggers a real-world interaction, e.g., starts a welding process or signals to a human that he/she can start working on a specific task.

Secondly, a feedback and control channel is required between the executing PSW and the controlling component. This means that there is the need for each service to register to an endpoint to report its status. This can be either the termination of its execution by correct completion (such as software calculation, or a welding operation accomplishment, or a human indication that the operation is done) or a report for the occurrence of an error. In the last case, our component can raise an exception, to signal the process execution engine to start the compensation mechanism, by obtaining a new PSP, avoiding the failing service.

Eventually, a common technical format for every service is required, to ease their deployment on cloud resources. As it is currently common practice to use containers [21], we also adopted this model. Our demonstrator uses the Docker Image format to represent every service grounding. This choice has the advantages of using an affirmed, widespread technical solution, which is also able to package all kinds of external resources within the image, supporting in this way our need for packaging heterogeneous services.

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

Thanks to this restricted set of three requirements, our system design is capable of integrating all kinds of services from the manufacturing domain smoothly. Additionally, these conditions are foundations to provide a real SOA abstraction of the existing manufacturing services.

In Figure 9, the UI of the OSL is shown. It can be used to monitor the status of deployed and running PSWs (in this case, a single service called "metal injection") and to stop their execution, if necessary.

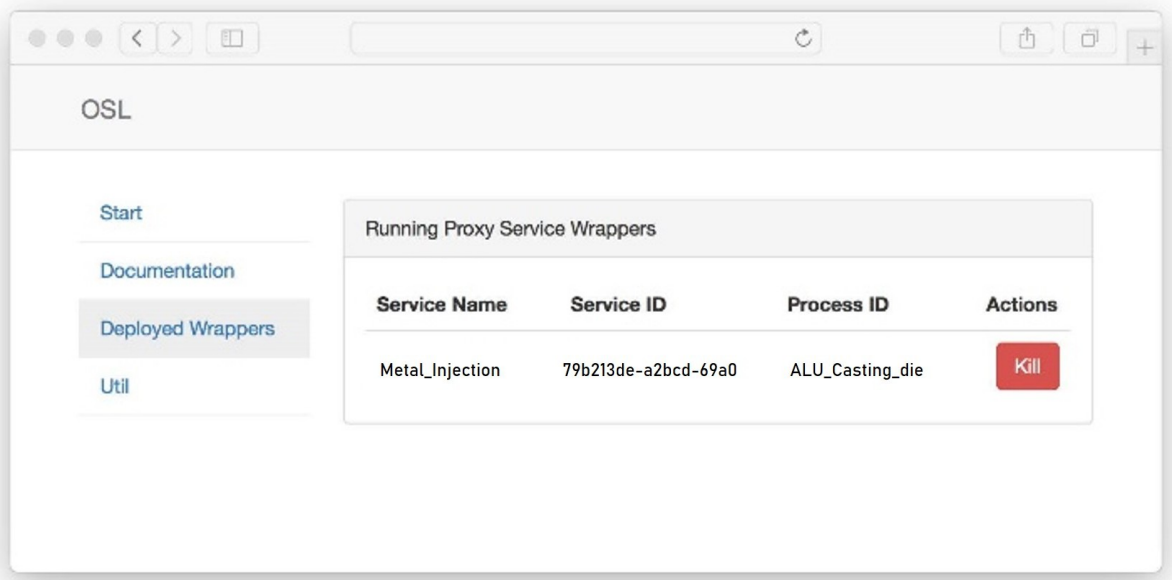

Figure 9. Screenshot of the OSL showing a deployed and running PSW called "metal injection", which is used in the "ALU casting die" process.

4. Demonstrative Application

To showcase the flexibility and usefulness of the proposed approach, we envisioned and designed a simple PM for the manufacturing process of bicycle bodies. It encompasses the injection of melted aluminum by all the processes required to prepare the cold-chamber molding machine and to control and expel the formed piece of hull.

As an initial step, we semantically described the services available to implement the different tasks in this manufacturing area. Figure 10 presents the results for three selected services, namely S_2, S_3 and S_4. This figure concentrates only on the semantic aspect, and for this reason, the other metadata to represent the grounding and the QoS measures for these services are not explicitly depicted.

Figure 11 depicts the BPMN, mainly a linear model composed of 11 tasks, three exclusive gateways (the first two defining together two alternative subpaths, to include or exclude T_2, whether the third is used to control a condition and repeat T_9 as many times as required). Each task is characterized by its IOPE annotation, as from Figure 5.

Interesting to note here is the fact that there is not always a perfect one-to-one correspondence between tasks and services, but services can be composed (e.g., S_{2+3} as the sequential arrangement of S_2 and S_3) or can be alternatively used (at least under certain assumption, such as the plugin compatibility) to implement the same task (e.g., T_2 in Figure 11 can be implemented by S_4 or by the composed service S_{2+3}). This is also part of the flexibility provided by a SemSOA approach.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 November 2018 doi:10.20944/preprints201810.0143.v2

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

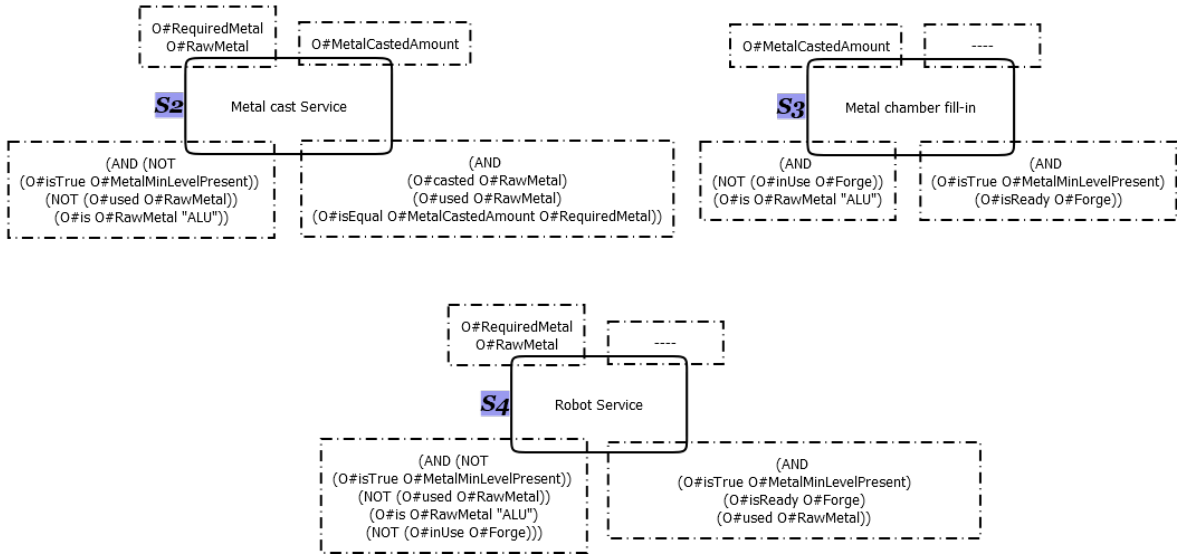

Figure 10. Some semantic services usable to implement Task 2 of the process model in Figure 11.

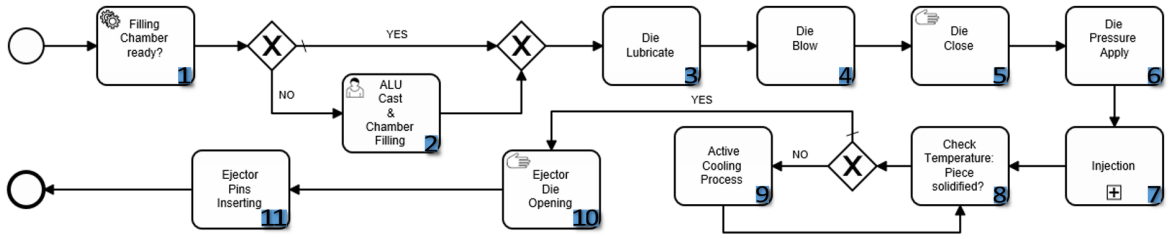

Figure 11. A process model for the forging of an aluminum hull for a bike body frame by injection.

In Listing 2, the optimization problem defined using our COP grammar for the PM in Figure 11 is presented: it starts (Line 2) by indicating the type of problem (linear and single objective), and then, it defines the variables (Line 5) , an array and two simple variables followed by many constants (Line 6) in both simple and array form. There are nine functions (Lines 8–16), ranging from a linear combination of variables (Lines 14–16) to the sum for the X array length between one or more parameters (Lines 8–10), passing through non-linear operators such as MAX and MIN (Lines 11–13).

Then, the constraints are presented (Lines 19–22): the first one limiting the sum of elements in X to the value of one (this, together with the domain definition for each entry in the set {0, 1} allows only one element in the X array to be not null); the second constraint devoted to guarantee that there is enough production time for the required batch. Then, a constraint on the electricity consumption is presented to assure it does not exceed the available amount during the production time; and eventually, one for securing that the marginal revenue level produced by the execution of the current batch of hulls is satisfactory, both with respect to its dimension and the average quarterly cash flow. To complete the problem class definition, the objective functions is stated, together with its association with the semantic concept (Line 24), to allow its reuse in composing the final service plan. In this case, the objective is to minimize **TotalCost(X)** for the production.

As presented before, the definition of the current instance of the problem is then given, starting from the domains of the variables (Line 27) and the values of the constraints (Lines 28, 29). Eventually, in section "INPUT", the mappings of semantic concepts (an extract of it is visible in Table 1) in the PM annotation are used to create the automatic binding of produced output to incoming input (Lines 31–46), based on the service QoS annotations.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 November 2018 doi:10.20944/preprints201810.0143.v2

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

Listing 2. The Constraint Optimization Problem (COP) associated with the model in Figure 11.

```
1  PROBLEM
2   TYPE linear single END TYPE
3   SOLVER both END SOLVER
4   CLASS
5   VARIABLES X[] VC Q END VARIABLES
6   CONSTANTS A1 A2 A3 B1 B2 B3 C1 C2 C3 Time[] Electricity[] ManteinTime[] SetupCost[] AmmCost[]
7          AcqTime[] Prec[] HoursAvaialble
8          END CONSTANTS
9   FUNCTIONS
10  Setup(T) = SUM(i,1,X.length, X[i] * SetupCost + X[i] * AmmCost[i])
11  ExecTime(T) = SUM(i,1,X.length, X[i] * Time[i])
12  ExecEletricity(T) = SUM(i,1,X.length, X[i] * Electricity[i])
13  ManteinanceTime(T) = MIN{X[i] * ManteinTime[i]}
14  AcquisitionTime(T) = MAX{X[i] * AcqTime[i]}
15  Precision(T) = MIN{X[i] * Prec[i]}
16  AvgTime(T) = A1 * ManteinanceTime(T) + A2 * ExecTime(T) - A3 * Precision(T)
17  ProdCostDirect(T) = B1 * ExecEletricity(T) - B2 * Precision(T) + B3 * Setup(T)
18  TotalCost(T) = C1 * AcquisitionTime(T) + C2 * AvgTime(T) + C3 * ProdCostDirect(T)
19  END FUNCTIONS
20  CONSTRAINTS
21  SUM(i,1,X.length, X[i]) = 1
22  AvgTime(T) <= HoursAvaialble / BatchSize
23  ExecEletricity(T) / BatchSize < MaxElectricity
24  QuaterlyCashFlow / QuaterlyProduction + MinQuaterlyRevenues > ProdCostDirect(T) * BatchSize
25  END CONSTRAINTS
26  minimize TotalCost(X) -> http://localhost/examples/Ont.owl#Cost
27  END CLASS
28  INSTANCE
29  DOMAINS X[]{0,1} VC[120.0,1275.0] Q[10.5,1000.0] END DOMAINS
30  VALUES A1 = 1.5 A2 = 0.2 A3 = 3 B1 = 7.1 B2 = 12.9 B3 = 1.55 C1 = 4.55 C2 = 7.75 C3 = 9.99
31         HoursAvaialble = ( DeliveryDate - StartProduction ) / WorkingDayHours - DeliveryTime
32  INPUT
33   BatchSize <- (Task_A, http://localhost/examples/Ont.owl#BatchDimension)
34   MaxElectricity <- (Task_A, http://localhost/examples/Ont.owl#ElectricitzySupplyCapabilities)
35   DeliveryDate <- (Task_A, http://localhost/examples/Ont.owl#DeliveryDate)
36   StartProduction <- (Task_A, http://localhost/examples/Ont.owl#StartProduction)
37   WorkingDayHours <- (Task_A, http://localhost/examples/Ont.owl#WorkingDayHours)
38   DeliveryTime <- (Task_A, http://localhost/examples/Ont.owl#DeliveryTime)
39   QuaterlyCashFlow <- (Task_C, http://localhost/examples/Ont.owl#QuaterlyCashFlow)
40   QuaterlyProduction <- (Task_C, http://localhost/examples/Ont.owl#QuaterlyProduction)
41   MinQuaterlyRevenues <- (Task_C, http://localhost/examples/Ont.owl#ExpectedQuaterlyRevenues)
42   Time <- (Task_F, http://localhost/examples/Ont.owl#ForgingTime)
43   Electricity <- (Task_F, http://localhost/examples/Ont.owl#ElectricityConsumption)
44   ManteinTime <- (Task_G, http://localhost/examples/Ont.owl#ManteinanceTime)
45   AcqTime <- (Task_B, http://localhost/examples/Ont.owl#BlockingServiceTime)
46   Prec <- (Task_F, http://localhost/examples/Ont.owl#ProductionPrecision)
47   SetupCost <- (Task_C, http://localhost/examples/Ont.owl#SetupCost)
48   AmmCost <- (Task_C, http://localhost/examples/Ont.owl#AmmortisationCost)
49  END INPUT
50  END VALUES
51  END INSTANCE
52  OUTPUT
53  VC -> (Task_F, http://localhost/examples/Ont.owl#VariableCost)
54  Q -> (Task_F, http://localhost/ontology/fake.owl#Quality)
55  END OUTPUT
56 END PROBLEM
```

The actual values used for comparing and contrasting services during the optimal PSP computation come from their QoS annotation, such as in Table 2. The final optional section "OUTPUT" instructs how to map back the found variables into the variable environment assignments (Lines 51, 52), which reflects how service input parameters are supposed to be set in order to yield the optimal objective values and the best level obtainable itself.

Table 1. The mapping between some QoS to provided semantic concepts, to support semantic type matching control during service plan generation and service invocation.The links point to publicly avaialble ontologies, pre–existjng to our work.

QoS Metric	Semantic Concept
ExecCost	http://localhost/examples/Ont.owl#Running_Cost
AmmCost	http://localhost/examples/Ont.owl#Amministrative_Cost
AcqCost	http://localhost/examples/Ont.owl#Acquisition_Cost
SetupCost	http://localhost/examples/Ont.owl#Setup_Cost
Time	http://www.w3.org/2006/time#TimePosition
Electricity	http://localhost/examples/CM_ontology.owl#Electric_Power
Precision (ppm)	http://purl.oclc.org/NET/ssnx/ssn#Precision
OutcomeRate	http://localhost/examples/Ont.owl#Production_Rate
DeliveryDate	http://localhost/examples/Ont.owl#DeliveryDate
BatchSize	http://www.owl-ontologies.com/mason.owl#batch_run_size
ManteinTime	http://www.w3.org/2006/time#Interval
MinQuaterlyRevenues	http://localhost/examples/Ont.owl#Expected_Revenues
StartProduction	http://www.w3.org/2006/time#TimePosition
AcqTime	http://www.w3.org/2006/time#Interval

Table 2. Subset of the QoS measured for the services S_2 and S_3 (respectively for melting the aluminum and for filling in the forming machine reservoir chamber till the expected level); and for service S_4, the robotic arm for the automatic refilling of the melted aluminum. These are alternatives to implementing Task 2 of the process model.

QoS Metric	Value		
	S_2	S_3	S_4
ExecCost	100	10	8
AmmCost	500	5	95
AcqCost	390	1	10,000
SetupCost	15	19	2390
Time	275	2	65
Electricity	575	15	1223
Precision (ppm)	1000	95	500
OutcomeRate	5	1255	2000

5. Results

At first, we defined the flow of invocation and data exchange at the base of the proposed demonstrator behavior. Figure 12 depicts the ordered sequence: at first, the user annotates (Step 1) the services using the provided UI; as a consequence, all the semantic services complete with metadata about QoS and the executable program, which offers the service (the so-called service grounding), are stored in a repository (Step 2).

Every dashed line color represents a different type of transferred object: light blue encodes for Semantic Services (SS), dark yellow for Process Models (PM), whereas green is for Process Service Plans (PSP). It is important to notice that models, instances and service plans were all encoded as BPMN v2.0-compliant XML documents, by the usage of extension elements: this allows maintaining in a single place the different stages of the models and storing them coherently in a unique repository. As for the notation in this representation, continuous lines represent explicit user actions, whereas fine dashed connections indicate implicit components' interaction to provide the service to the user. In Step 3, the process model is designed, together with the IOPE characterization and all the metadata required and stored in the central extended BPMN repository (Step 4). This finishes the preparation, until the moment the user is ready to execute an instance of a defined process model. It is important

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

to notice that this approach supports the theoretical separation between the service annotator, the process model designer and the operator in charge of executing the instance of this manufacturing process. In Step 5, through the RunTime execution environment, the human operator launches an instance of the process, causing it to retrieve the model from the repository (Step 6) and passing it to the optimization component (Step 7). After retrieving the set of available semantic services (Step 8), the optimization component computes a non-dominated process service plan by functional composition and non-functional optimization of the related COP definition and returns it to the RunTime execution component (Step 10). Furthermore, the optimization component stores the plan in the eBPMN archive facility (Step 9).

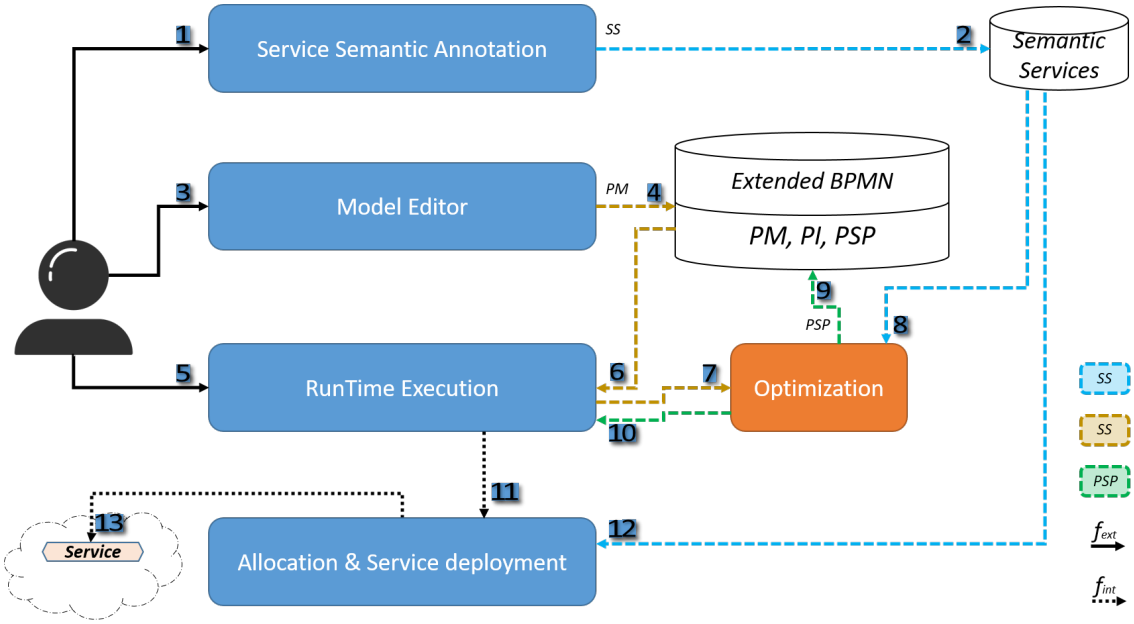

Figure 12. The flow of interaction and data exchange between the components of the proposed solution. The data exchange is as follows: light blue dashed line for Semantic Services (SS), dark yellow for Process Models (PM) and green for Process Service Plans (PSP). Black continuous arrows indicate user function call, and black dashed ones pinpoint internal function calls.

The process then continues without user interaction, as the RunTime execution component analyzes the produced process service plan one service at a time, in the corresponding order for the process model. Here, the first service, equivalent to T_1 in Figure 11, is considered, and the service metadata are passed in Step 11 to the allocation and service deployment together with all the environmental and contextual variables and settings necessary for the service. Eventually, that enactment component retrieves from the repository the full information set of the service (Step 12) and, after initializing the environment, deploys its grounding in the cloud (Step 13), monitoring it and collecting the returned value(s). Once the service has finished its task, it is also necessary to dispose of the deployment wrapper and environment, in order to release the cloud resources used by this service. If the current service is correctly executed, the allocation and service deployment component interacts with the RunTime execution to move to the next service used to implement the model, till the full process service plan is completed, and the UI returns a positive confirmation of the termination to the user.

Anyway, it is possible that a service is unable to be deployed (such as when it is a physical resource already in use or out-of-service for unplanned maintenance) or a failure code is returned. Figure 13 represents this case: the first task was terminated correctly by the execution of S_1, then the PSP dictated to use S_4 for implementing T_2, as this was the best match (amongst the semantically-quasi-equivalent

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 November 2018 doi:10.20944/preprints201810.0143.v2

Peer-reviewed version available at Information 2018, 9, 279; doi:10.3390/info9110279

alternatives $\{S_{2+3}, S_4\}$ from Figure 10), but the robotic arm used to ground S_4 is currently unusable, making the service failing.

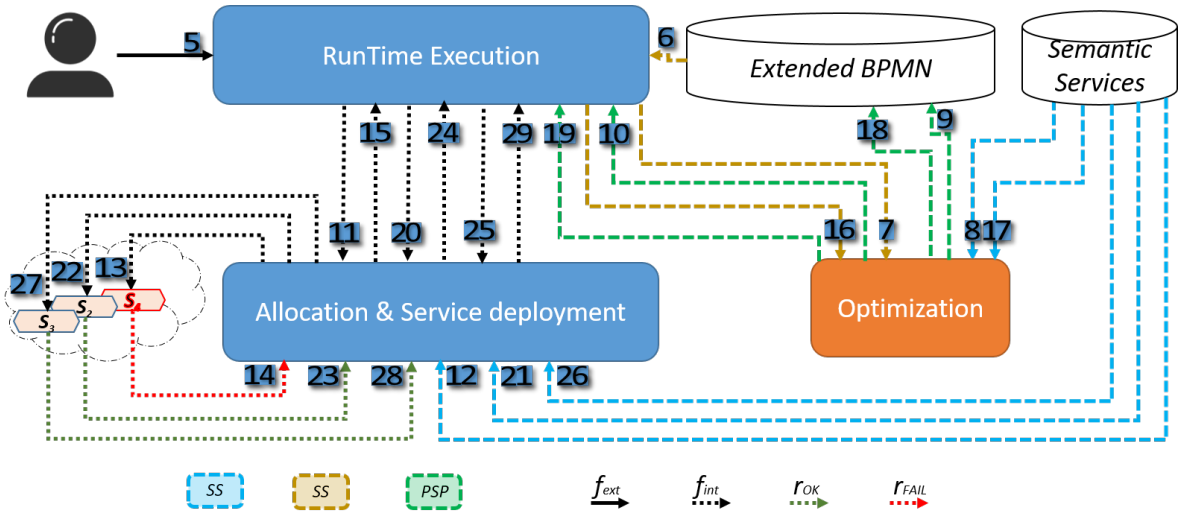

Figure 13. Whenever a requested service became unavailable, the proposed architecture allows for fast and incremental replanning. Here, service S_4's deployment in the cloud failed (red dashed arrow), and CREMA proposed to achieve T_2 objectives by using the service combination $S_2 + S_3$, which constitutes a semantic equivalent (valid under plugin relaxation) candidate. They are successful (green dashed arrows), and this fulfills the requirements for T_2's correct execution in the PM.

The interaction flow follows the same evolution as for the "none failure" case until Step 13. In Step 14, the service control returns a failure status to the calling component. This triggers back the RunTime execution environment (Step 15), which collects all the information about the services correctly executed and the failing ones for the current process (*PSP log*), and it requests an alternative PSP (Step 16) via the optimizer. Using the additional knowledge about the execution log and the services currently available (Step 17), a new functional composition and a new QoS-based COP resolution are computed. This generates a viable non-dominated PSP, stored (Step 18) and subsequently returned to the execution component (Step 19), which is then able to resume the process execution, without the need for aborting the already executed services. This is particularly critical in the manufacturing domain, as most part of the services is not-idempotent and cannot be repeated without affecting the final result or generating scrap and defective parts. If there are no possible services available to implement the process, the user will observe a failure ("broken" PSP). In this demonstrative case, we suppose there is a combination that can be used as functionally equivalent for T_2, even though it is sub-optimal with respect of the COP objective and the QoS measures for the services.

After that, the normal iteration is restored between the execution and the deployment component, using the new PSP, which has an updated grounding for T_2, composed by S_2 and S_3. Each single service is then individually instantiated: Steps 20 and 25 report the service deployment request; Steps 21 and 26 pinpoint the service details retrieved from the repository; Steps 22 and 27 show the actual deployment of the service in the complete and correct cloud environment; Steps 23 and 28 signal the correct completion of the service execution; whereas eventually, Steps 24 and 29 confirm it for the RunTime execution environment, in order to allow it to proceed with the next service defined by the PSP. Eventually, when the last service is correctly returned, the execution of the process instance is done, and the human operator is informed through a message in the UI.

6. Discussion

In this section, we present our thoughts about the effort and major barriers for the extension of the presented demonstrative case to general manufacturing-related processes, towards a pragmatic Industry 4.0-enabled approach in physical production contexts. The analysis is divided following the presented components of our demonstrator.

As a base, we proposed a basic general domain ontology, and we showcased how it can be treated and extended to cover the specific sub-domain for a new application. We showed that it is feasible to implement a semantic wrapper for the service annotation; and that it is possible to use it for representing whatever type of service (information retrieval or computation, mechanical/operative tool or robot and human operators) in a XaaS approach. This wrapping capability showed itself able also to equip the service description with user-defined QoS. These steps are still a considerable obstacle for process designers, as they are not normally familiar with domain knowledge elicitation and its formalization. Additionally, the need for defining the semantic and metadata (QoS) wrapper around all the available service, at the finest granularity possible, is also a big barrier for the adoption of XaaS in manufacturing. On the positive side, once these demanding and challenging tasks are done, it normally should not be necessary to repeat them, as they can support every process in the defined sub-domain. Unfortunately, in reality, both small corrections and, sometimes, big reworks are occasionally necessary, to better focus on or to correct a misunderstanding in these element settings.

Minimal modifications are necessary on the BPMN editor side, to support the extension in the standard language for including the COP problem description; the service plan, the data bindings and the optimal variable assignment; as well as the execution log. We exposed that this is already enough to support the expected functionalities. Designing extended BPMN models equipped with COP definition and tasks' semantic annotations should not be too challenging, once the previous two steps are internalized. For the rest, these are a standard model following the well-known notation, and their graphical representation follows the usual aspect familiar to the process designer.

From the point of view of the processes optimization, the main benefits with respect to the existing approaches are manifold: the first improvement is the business process formulation, as it allows the full integration of functional service selection and composition with non-functional optimization based on user-defined QoS and objective functions arbitrarily complex in the COP. This is achieved through our BPMN extensions and thanks to the development of a grammar for the optimization problem formalization. Secondly, the produced output, the PSP, is directly enactable by an execution environment, being a complete plan. This means that it is equipped with all the relevant information: service assignments, data flow (variable bindings) and optimal variable assignments for initializing the enactment environment. Eventually, by encoding the computed PSP in an extended BPMN format, it allows maintaining in a single place the model and plan, together with the variables' assignment and the optimality value achieved. This part is completely transparent to the process designer and the human operator in charge of supervising the process execution, except for the approval of the proposed service plan, for the responsibility assumption.

Regarding the execution part, thanks to a small interpretation effort for the model/instance additional fields, the main advantage is the capability of using the optimal PSP and data binding information, together with the decoupling of the BPMN structure (conditionals and gateways) from the service deployment, which is delayed and demanded by a specialized component.

Regarding the service execution and control, our specialized component is a complete novelty in this domain and allows checking the status (availability and leasability) of any service in the form of a container and to support late deployment on cloud facilities. Furthermore, these last two components work in the background of the demonstrator, except for the very standard UI they provide for controlling the process execution and the service deployment, so this should not represent any barrier for the practical adoption of such an approach in real-world manufacturing industries.

7. Conclusions

In this work, we presented our innovative pragmatic solution for an Industry 4.0 application in the manufacturing domain. It is based on formalized domain knowledge and structured service wrapped with semantic annotation, to provide dynamic and just-in-time process plans. After introducing the ontology and the service annotation tool, we concentrated on presenting the optimization component, which composes functionally-correct plans and supports the optimization of non-functional aspects, in the form of a COP, using as measures generic QoS and supporting user-defined composed objective functions. Then, we depicted the role of the execution tool and the service deployment facility, indicating how they make use of the computed optimal process service plan for enacting in the cloud the actual service grounding, producing in this way the expected results of the model.

To showcase the capabilities of the tool, we applied it to a scenario in the manufacturing domain. Our tool combination allowed practically transposing a real-world process for aluminum forging by injection into a fully-functioning Industry 4.0-enabled process. The main point of this demonstrator is to showcase that it is currently possible to implement an existing structured manufacturing practice into an optimized ICT-supported process, taking advantage of the flexibility and effectiveness of dynamic service binding and deployment.

Regarding the extensibility of this demonstrator and the barriers for its adoption, we identified that they are concentrated mainly in the initial phases of such an approach, namely in the domain knowledge elicitation, its formalization into an ontology (maybe an extension of our current proposal) and in the elementary service identification and annotation with semantics and the QoS metrics value. Fortunately, these complex and highly valuable kick-off activities can be executed once and can be highly supported by external competence from semantic and SOA experts, maybe in the form of a consultant for the company's innovation. For all the other parts of the proposed demonstrator, despite its minimal coverage of all the possibilities sketched by the Industry 4.0 trends existing in the literature, we perceive that the modifications will be mainly limited to the ICT infrastructure and software adopted by the manufacturing enterprise. All the other adjustments needed in the operative practice of the corporation can be over-compensated by the flexibility and dynamicity naturally provided by the proposed approach.

Author Contributions: Conceptualization, L.M. Methodology, L.M. Software, P.K., P.W., and L.M. Data curation, L.M. Writing, original draft preparation, L.M. Writing, review and editing, L.M. and P.W. Visualization, L.M. and P.K. Supervision, L.M. Project administration, M.K.

Funding: This work was partially financed by the European Commission within the H2020-RIAproject CREMA, Agreement No. 637066; by the German Federal Ministry of Education and Research (BMBF) within the project INVERSIV; and by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

Acknowledgments: The authors would like to thank all the CREMA project partners (please refer to http://www.crema-project.eu/partners/) for the comments and contributions to the ideas behind the realized solutions. For the software realization, we would like also to publicly thanks Michael Borkowski, Christoph Hochreiner and Olena Skarlat from the Distributed System Group at Technische Universität Wien (TU—Wien).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

Peer-reviewed version available at *Information* **2018**, *9*, 279; doi:10.3390/info9110279

ODERU	Optimisation at Design tim e and Run-time component
DFKI	Deutsches Forschungszentrum für Künstliche Intelligenz (German Research Center for Artificial Intelligence)
XaaS	Everything as a Service
SOA	Service-Oriented Architecture
semSOA	Semantic SOA
OWL2	Web Ontology Language, Release 2
OWL-S	OWL Web-Service description
WSDL	Web Service Description Language
BPMN	Business Process Modeling Notation
BPEL	Business Process Execution Language
BPMO	Business Process Modeling Ontology
sBPMN	Semantic BPMN
IOPE	Inputs, Outputs, Preconditions and Effects
QoS	Quality-of-Service
COP	Constrained Optimization Problem
IoT	Internet of Things
PM	Process Model
PI	Process Instance
PSP	Process Service Plan
CREMA	Cloud-based Rapid Elastic MAnufacturing
muCOP	multi-objective COP
WSML	Web Service Modeling Language
SA-WSDL	Semantic Annotations for the WSDL and XML Schema
USDL	Unified Service Description Language
VM	Virtual Machine
SLA	Service Level Agreement
CDM	CREMA Data Model
antlr	ANother Tool for Language Recognition

References

1. Bayo-Moriones, A.; Billón, M.; Lera-López, F. Perceived performance effects of ICT in manufacturing SMEs. *Ind. Manag. Data Syst.* **2013**, *113*, 117–135.
2. Jovane, F.; Westkämper, E.; Williams, D. *The ManuFuture Road: Towards Competitive and Sustainable High-Adding-Value Manufacturing*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2008.
3. Taisch, M.; Stahl, B.; Tavola, G. ICT in manufacturing: Trends and challenges for 2020—An European view. In Proceedings of the 2012 10th IEEE International Conference on Industrial Informatics (INDIN), Beijing, China, 25–27 July 2012; pp. 941–946.
4. Stock, T.; Seliger, G. Opportunities of sustainable manufacturing in industry 4.0. *Procedia Cirp* **2016**, *40*, 536–541.
5. Weske, M. Business process management architectures. In *Business Process Management*; Springer: Berlin/Heidelberg, Germany,2012; pp. 333–371.
6. Weyer, S.; Schmitt, M.; Ohmer, M.; Gorecky, D. Towards Industry 4.0-Standardization as the crucial challenge for highly modular, multi-vendor production systems. *Ifac-Papersonline* **2015**, *48*, 579–584.
7. McIlraith, S.A.; Son, T.C.; Zeng, H. Semantic web services. *IEEE Intell. Syst.* **2001**, *16*, 46–53.
8. Duan, Y.; Fu, G.; Zhou, N.; Sun, X.; Narendra, N.C.; Hu, B. Everything as a service (XaaS) on the cloud: origins, current and future trends. In Proceedings of the IEEE 8th International Conference on Cloud Computing (CLOUD), New York, NY, USA, 27 June–2 July 2015; pp. 621–628.
9. Hepp, M.; Leymann, F.; Domingue, J.; Wahler, A.; Fensel, D. Semantic business process management: A vision towards using semantic web services for business process management. In Proceedings of the

2005 IEEE International Conference on e-Business Engineering (ICEBE), Beijing, China, 12–18 October 2005; pp. 535–540.

10. Abramowicz, W.; Filipowska, A.; Kaczmarek, M.; Kaczmarek, T. Semantically enhanced business process modeling notation. In *Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications*; IGI Global: Hershey, PA, USA; 2012; pp. 259–275.
11. Dimitrov, M.; Simov, A.; Stein, S.; Konstantinov, M. A BPMN based semantic business process modelling environment. In Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007), Innsbruck, Austria, 7 June 2007; Volume 251.
12. Born, M.; Hoffmann, J.; Kaczmarek, T.; Kowalkiewicz, M.; Markovic, I.; Scicluna, J.; Weber, I.; Zhou, X. Semantic annotation and composition of business processes with Maestro for BPMN. In Proceedings of the 2008 European Semantic Web Conference, Tenerife, Spain, June 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 772–776.
13. Karastoyanova, D.; van Lessen, T.; Leymann, F.; Ma, Z.; Nitzche, J.; Wetzstein, B. Semantic Business Process Management: Applying Ontologies in BPM. In *Handbook of Research on Business Process Modeling*; IGI Global: Hershey, PA, USA, 2009; pp. 299–317.
14. Weber, I.; Hoffmann, J.; Mendling, J. Beyond soundness: On the verification of semantic business process models. *Distrib. Parallel Databases* **2010**, *27*, 271–343.
15. Klusch, M.; Kapahnke, P.; Schulte, S.; Lecue, F.; Bernstein, A. Semantic web service search: A brief survey. *KI-Künstliche Intell.* **2016**, *30*, 139–147.
16. Pilioura, T.; Tsalgatidou, A. Unified publication and discovery of semantic web services. *ACM Trans. Web* **2009**, *3*, 11.
17. Zhang, Y.; Huang, H.; Yang, D.; Zhang, H.; Chao, H.C.; Huang, Y.M. Bring QoS to P2P-based semantic service discovery for the Universal Network. *Pers. Ubiquitous Comput.* **2009**, *13*, 471–477.
18. Klusch, M.; Gerber, A. Fast composition planning of OWL-S services and application. In Proceedings of the 2006 4th European Conference on Web Services, Zurich, Switzerland, 4–6 December 2006; pp. 181–190.
19. Strunk, A. QoS-aware service composition: A survey. In Proceedings of the 2010 IEEE 8th European Conference on Web Services (ECOWS), Ayia Napa, Cyprus, 1–3 December 2010; pp. 67–74.
20. Zou, G.; Lu, Q.; Chen, Y.; Huang, R.; Xu, Y.; Xiang, Y. QoS-aware dynamic composition of web services using numerical temporal planning. *IEEE Trans. Serv. Comput.* **2014**, *7*, 18–31.
21. Bernstein, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84.
22. Thoenes, J. Microservices. *IEEE Softw.* **2015**, *32*, 116.
23. Slominski, A.; Muthusamy, V.; Khalaf, R. Building a multi-tenant cloud service from legacy code with docker containers. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E), Tempe, AZ, USA, 9–12 March 2015; pp. 394–396.
24. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 171–172.
25. Schulte, S.; Janiesch, C.; Venugopal, S.; Weber, I.; Hoenisch, P. Elastic business process management: state of the art and open challenges for BPM in the cloud. *Future Gener. Comput. Syst.* **2015**, *46*, 36–50.
26. Schulte, S.; Hoenisch, P.; Venugopal, S.; Dustdar, S. Introducing the Vienna Platform for Elastic Processes. In Proceedings of the 10th International Conference on Service Oriented Computing, Shanghai, China, 12–15 November 2012; Volume 7759, pp. 179–190.
27. Hoenisch, P.; Hochreiner, C.; Schuller, D.; Schulte, S.; Mendling, J.; Dustdar, S. Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds. In Proceedings of the 8th International Conference on Cloud Computing, New York, NY, USA , 27 June–2 July 2015; pp. 17–24.
28. Hoenisch, P.; Schuller, D.; Schulte, S.; Hochreiner, C.; Dustdar, S. Optimization of Complex Elastic Processes. *Trans. Serv. Comput.* **2016**, *9*, 700–713.
29. Juhnke, E.; Dörnemann, T.; Bock, D.; Freisleben, B. Multi-objective Scheduling of BPEL Workflows in Geographically Distributed Clouds. In Proceedings of the 4th International Conference on Cloud Computing, Washington, DC, USA, 4–9 July 2011; pp. 412–419.
30. Wei, Y.; Blake, M.B. Proactive virtualized resource management for service workflows in the cloud. *Computing* **2014**, *96*, 1–16.

31. Bessai, K.; Youcef, S.; Oulamara, A.; Godart, C. Bi-criteria strategies for business processes scheduling in cloud environments with fairness metrics. In Proceedings of the 7th International Conference on Research Challenges in Information Science, Paris, France, 29–31 May 2013; pp. 1–10.
32. Cai, Z.; Li, X.; Gupta, J.N. Critical Path-Based Iterative Heuristic for Workflow Scheduling in Utility and Cloud Computing. In Proceedings of the 11th International Conference on Service Oriented Computing, Berlin, Germany, 2–5 December 2013; Volume 8274, pp. 207–221.
33. Chen, C.C.; Lin, Y.C.; Hung, M.H.; Lin, C.Y.; Tsai, Y.J.; Cheng, F.T. A novel cloud manufacturing framework with auto-scaling capability for the machining industry. *Int. J. Comput. Integr. Manuf.* **2016**, *29*, 786–804.
34. Mazzola, L.; Kapahnke, P.; Vujic, M.; Klusch, M. 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, Porto, Portugal, 9–11 November 2016; Volume 2, pp. 136–143.
35. Mazzola, L.; Kapahnke, P. DLP: A Web-based Facility for Exploration and Basic Modification of Ontologies by Domain Experts. In Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services (iiWAS '17), Salzburg, Austria, 4–6 December 2017; ACM: New York, NY, USA, 2017; pp. 446–450.
36. Burstein, M.; Hobbs, J.; Lassila, O.; Mcdermott, D.; Mcilraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; Sirin, E.; et al. OWL-S: Semantic markup for web services. *W3C Memb. Submiss.* **2004**, 22(4).
37. Mazzola, L.; Kapahnke, P.; Klusch, M. Semantic Composition of Optimal Process Service Plans in Manufacturing with ODERU. *Web Inf. Syst.* **2018**, *14(4)*.
38. Mazzola, L.; Kapahnke, P.; Klusch, M. ODERU: Optimisation of Semantic Service-Based Processes in Manufacturing. In Proceedings of the International Conference on Knowledge Engineering and the Semantic Web, Szczecin, Poland, 8–10 November 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 337–346.
39. Klusch, M.; Kapahnke, P. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semant. Sci. Serv. Agents World Wide Web* **2012**, *15*, 1–14.
40. Klusch, M. Overview of the S3 contest: Performance evaluation of semantic service matchmakers. In *Semantic Web Services*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 17–34.
41. Rodriguez-Mier, P.; Pedrinaci, C.; Lama, M.; Mucientes, M. An integrated semantic Web service discovery and composition framework. *IEEE Trans. Serv. Comput.* **2016**, *9*, 537–550.
42. Mazzola, L.; Kapahnke, P.; Waibel, P.; Hochreiner, C.; Klusch, M. FCE4BPMN: On-demand QoS-based Optimised Process Model Execution in the Cloud. In Proceedings of the 23rd ICE/IEEE ITMC Conference, Madeira, Portugal, 27–29 June 2017.