Cloud-based Infrastructure for Workflow and Service Engineering Using Semantic Web Technologies

Volkan Gezer and Simon Bergweiler

German Research Center for Artificial Intelligence (DFKI)
Innovative Factory Systems
Kaiserslautern, Germany
Email: firstname.lastname@dfki.de

Abstract—**This paper presents the concept and implementation of a cloud-based infrastructure platform and tailored tools for graphical support for user-driven experiments. The focus lies on the creation of a platform that allows multiple users to integrate their expertise via Web service interfaces and combine them to compose workflows for their experiments in the engineering domain. The platform employs Semantic Web technologies, which increase interoperability of the services and assist during workflow design by suggesting compatible services. The implemented tools give users the possibility to utilize the platform using a standard Web browser, without any knowledge of service engineering and the underlying complex technologies. An experiment is described as a workflow and consists of orchestrated services from several software vendors that encapsulate specific tasks for improving product development. One advantage of this approach is the automatic execution of the services and data flow among them. The cloud-based platform can also be combined with high-performance computing when services require complex calculations. The created platform offers optimal conditions to involve independent specialists and conduct short or long-running experiments, depending on the complexity of the task. This results in tremendous time savings and allows experts to carry out more experiments with products, which were omitted due to the complexity and the limited computing power, until now. The possibility to conduct these experiments improves the productive know-how of the companies and enhances the products they are selling.**

Keywords–Cloud infrastructure; semantic workflow description; Semantic Web services; graphical workflow editing; workflow execution.

I. Introduction

In this paper, we present the concept and implementation of a flexible cloud-based platform for the vendor-independent integration of Semantic Web services and their execution in the engineering domain. This platform is provided as Infrastructure as a Service (IaaS), and is able to integrate, combine and orchestrate Web services [1]. Cloud-based solutions are part of the daily life, and their usage is increasing day by day. The advantage of access to data from a cloud solution from anywhere allows increased mobility of people and their applications, and changes also behavior and attitude of responsible persons in the engineering domain [2].

Involving Semantic Web technologies inside a cloud-based solution significantly improves usability by structuring the data in a standardized way. These standardized data structures can be understood by machines and humans and utilized to create interoperable and vendor-independent applications. With this approach vendor lock-in problems can be avoided [3].

The platform enables experts from various application domains to independently plan, design, and execute their individual experiments for the analysis and optimization of their products. Each experiment is described as a workflow that uses the functionality of different products of different software vendors in any combination. With the help of this platform, several independent specialists act as software vendors and offer their expertise, e.g., strong calculation procedures or routines for the comparison of 3D models, wrapped by Web service interfaces in a system-wide infrastructure. These advanced analytical capabilities, which are accessible via the platform, can be used to enhance the products, identify weaknesses and subsequently improve the positive effects of the products.

Thanks to this distributed architecture, the platform offers optimal conditions for both short and long-running experiments [4]. To provide an added value and according to customer requirements, the developed platform is able to pass the execution of dedicated services to a cluster of high-performance computers (HPC). These HPC clusters are to perform calculation of complex tasks and are spread across different virtualization solutions. This design of the platform allows experts to carry out more experiments with products because of enormous time savings. The developed tailored tools of this cloud-based platform, described below as core backend components, allow engineering companies and software solution vendors to integrate their Software as a Service (SaaS). Services are orchestrated in specific workflows, seamlessly supported by graphical user interfaces. They do not even require specific skills or knowledge of the underlying Semantic Web technologies. The developed solution uses standardized Internet technologies and all workflows can be executed using a standard Web browser, requiring no additional software download and installation. The provided platform wraps all complexity of the technologies and provides Application Programming Interfaces (API) for communication.

Section II introduces used technologies and describes the topics under consideration. For a better understanding, Section III describes a concrete application scenario and the requirements in the engineering domain. This leads over to Section IV, where the methodology and concept of the developed approach is discussed. Section V describes the architecture and the interaction of the developed core backend components. The paper ends with a conclusion and an outlook on future work and extensions.

II. Background

A set of complementary reusable functionalities that are provided for various purposes by software are called “services”. If a service is offered via World Wide Web using Web technologies, such as the Hypertext Transfer Protocol (HTTP), the service is labeled as Web service. Web services are designed to support machine-to-machine interaction over a network and allow interoperable communication [5]. With the help of description languages, which will be discussed in the upcoming sections, Web services create communication between peer-platforms, prevent vendor dependency and increase reusability.

A. Web Services Description Language

The Web Service Description Language (WSDL) is a language and platform-independent XML-based interface definition language, designed with the aim to create a standardized mechanism for the description of Web services. It describes SOAP-based Web services in detail, their technical input and output parameters, ports, data types, and how services must be invoked. With this machine-readable description language, the automatic detection and execution of Web services is possible. A ready-revised language draft was submitted to the World Wide Web Consortium (W3C) [6], but only version 2.0 was standardized and proclaimed as W3C recommendation [7]. Unfortunately, WSDL is a lower level interface description language that addresses the technical mechanisms and aspects of Web services, and it does not reflect the functionality of a service. Furthermore, it is difficult to create and understand for humans. In this approach, WSDL is used for the technical description of Web services, their input and output parameters, and the SOAP messaging mechanism.

B. Technologies of the Semantic Web

The development of the current Web to the “Semantic Web” is pervasive. Efforts are aiming to add annotations to things and objects of daily life. Through the help of annotations, the vision of the Semantic Web allows better cooperation between people and computers; well-defined meanings are attached to information [8]. The Resource Description Framework (RDF) is one of the most important data formats that has been developed to implement this vision. The Semantic Web combines technologies that deal with the description of information and knowledge sources, such as ontologies, RDF triple stores, and Semantic Web services [9][10]. Ontologies allow the definition of a vocabulary of a dedicated application domain and define for this purpose concepts and properties. These concepts can in turn be connected by relations, which promise a significant value, when conclusions are drawn about these structures. In that field, the W3C defines its recommendations as an open standard like RDF(S) [9][11] and the Web Ontology Language (OWL) [12].

In contrast to a complex and comprehensive infrastructure that tries to solve all problems of the interaction and communication of distributed applications, the Semantic Web Technology Stack, depicted in Figure 1, is a family of modular standards mostly standardized by the W3C. Each of these standards aims at another part of problem or another sub-problem.

This stack of Semantic Web technologies describes the vision of the W3C to create a Web of linked data. The idea of open data stores on the Web, the ability to build vocabularies, and write rules for handling data based on these empowered technologies, such as RDF, OWL, and the SPARQL Protocol And RDF Query Language (SPARQL) [13].

In the following, the individual layers and the associated standards will be briefly explained. Starting at the lowest level with the Unicode standard and Uniform Resource Identifiers (URI). The Unicode standard allows an unambiguous name

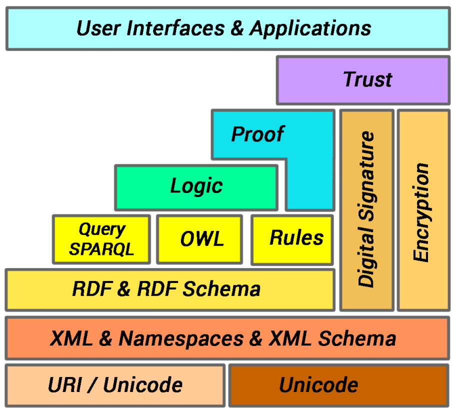


Figure 1. Semantic Web technology stack.

of the resources in any language. This makes the approach multilingual and forces the exchange of messages. In order to identify resources on the Web, it is necessary to have URIs and to use them for the concrete description of specific resources. URIs are used to describe a specific resource uniquely. XML forms the central core technology and is added to the second level of this architecture or hierarchy. XML is used for the storage and universal exchange of data. The technologies on the upper layers rely on it. The third layer of the Semantic Web technology stack is probably the most important layer of all. RDF makes it possible to create metadata in machine-readable form. Here, a sentence structure in the form of triples (subject, predicate, and object) is used. The layers three and four are important pillars of the Semantic Web, they host RDF, RDF Schema, ontologies, and their vocabulary. They serve the communication of different, mutually independent domains at the semantic level. The uppermost layers of the architecture are logic, proof and trust. The Logic level provides technologies to allow computers to recognize specific patterns by the help of dedicated rules. By modifying this rule structures and patterns, new knowledge is inferred and exploited. The Proof level should make it possible to distinguish the trustworthiness of resources. Together with the Digital Signature layer it is possible that computers are able to cope trusting tasks without human intervention.

C. Semantic Web Services

In the recent years, the tendency towards Semantic Web technologies increased the research in the domain, results in an elevated number of available ontologies as well as standards recommended by the W3C. To widen the scope of applicability, one of the submitted ontologies to W3C was the Web Ontology Language for Web Services (OWL-S), which allowed services on the Web to be found, executed, and monitored. The OWL-S ontology is designed on top of OWL with extensions to make service discovery, invocation, composition, and monitoring possible. The provided structure also allowed these operations to be performed autonomously, when desired [14]. Based on the previously described technologies, domain models must be created to form an important conceptual basis. Therefore, parts of the dedicated knowledge domain are categorized and structured in a machine readable form. OWL-S [14] extends this base to a set of constructs that relate to properties, specialties and dependencies of the Web service level and is also machine readable and processable.

A concrete service description in OWL-S is separated in several parts. Figure 2 shows the main concepts and relations of a service model in OWL-S: service profile, service model, service grounding, and important for our approach, the processes.

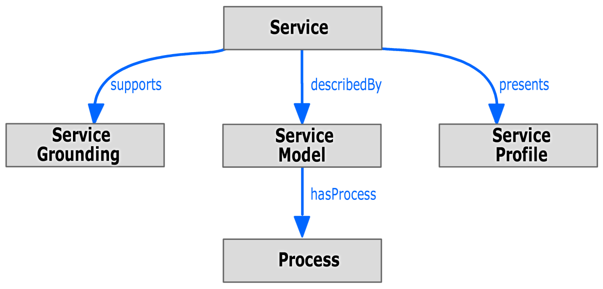


Figure 2. Main Web service concepts in OWL-S.

OWL-S API provides its own native engine to execute Web services described only in WSDL 1.2 and consequently, SOAP interface. Any Web services provided with a WSDL description and SOAP interface can be integrated into the infrastructure and converted into semantic Web service descriptions as long as they satisfy the requirements. For a Web service that is converted into a Semantic Web service with the help of an upper ontology, the following required types of the OWL-S Submission [14] are created:

- The *Service Profile* provides information to describe a service to a requester. The profile provides three types of information: the service creator, the service functionality, and the service characteristics [14]. It is also is used for service discovery and describes the functionality of the service and contains information about the service provider. Furthermore, this profile reflects the overall functionality of a service with its precondition, input and output types, features, and benefits.
- The *Service Model* is a mandatory type for the description how a Web service works. The model describes the inputs, outputs, preconditions and effects. It also specifies the *Process* concepts and their execution order. The process description consists of simple atomic processes or complex composite processes that are sometimes abstract and not executable. Each function provided by the service is considered as an *Atomic Process*, whereas combined multiple services are named as *Composite Service*.
- The *Service Grounding* stores the detailed technical communication information on protocols and formats. This concept provides the physical location to the technical description realized in WSDL. This WSDL-file is called when the service is executed, as well as during conversion process to retrieve the technical inputs and outputs of the service.

The described core concepts are shown in detail in Figure 3 with their subtypes and processes. The listed types provide the basis for OWL-S to create and use relations, which are utilized for improved interoperability.

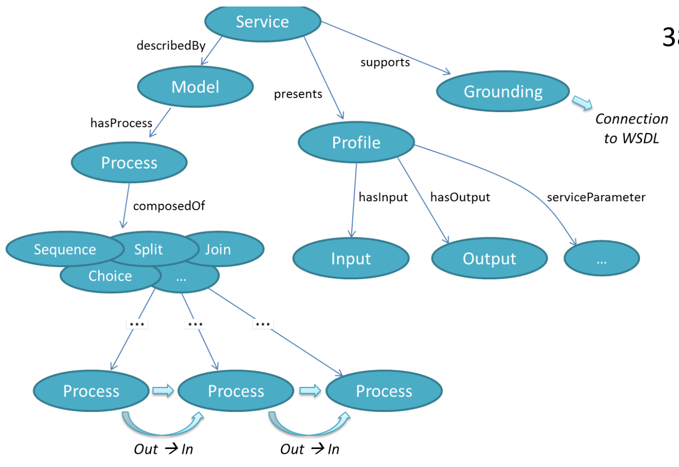


Figure 3. Upper service ontology for OWL-S.

However, the relations generated using only OWL-S ontology form the minimal relationship for services, enough to be operated. If the usage scenario requires involvement of additional relations, these must be defined creating a *service and workflow ontology* and including it inside a Semantic Repository [15]. This ontology can contain more contextual information with relations to enhance the interoperability [16].

D. Triple Stores

Computational tasks often require collection and storage of the results for further usage. Storage of information without a structured form increases the complexity and the time to access the data, and reducing the flexibility for further modifications and enhancements [17] causing fragmentation problems. To address this issue, databases play the role as containers, which collect and organize the data for swift future access [18].

Semantic repositories are similar to the database management systems (DBMS) in terms of providing functionality for organization, storage, and querying the data, but differ from them in terms of the type of organization and data representation. Unlike DBMS, semantic repositories use schemata to structure the data, but are also able to establish relationships between stored values. Regarding to data representations, semantic repositories work with flexible and generic physical data models, which allow merging other ontologies "on the fly" and relate the data among merged schemata [19]. As OWL-S is based on OWL, which is built on top of RDF, see Section II-B, the data operations are performed using the same RDF structure. This structure provides descriptions to query the data, and allows optimal extension of relations allowing multiple use. The Sesame framework [20] is one RDF storage solution, which can be used in this context. It allows creation, processing, editing, storing, and querying RDF data, therefore, it is chosen to serve as a storage solution in this approach.

E. Business Process Execution Language and OWL-S

The Business Process Execution Language (BPEL) is a language for describing and executing business processes in general. It provides an XML-based syntax and allows data manipulation for data processing and data flow. It also allows orchestration of services, after specifying the service set and the service execution order [21]. Numerous platforms (such as Oracle BPEL Process Manager, IBM WebSphere Application Server Enterprise, IBM WebSphere Studio Application Developer Integration Edition, and Microsoft BizTalk Server 2004) support the execution of BPEL processes. Some of these platforms also provide graphical editing tools.

For the languages OWL-S and BPEL, there exist tools for the automated execution of Web services described in WSDL. They also permit implementations in any programming languages as long as they provide valid WSDL descriptions. Different from BPEL, OWL-S facilitates Semantic Web technologies, which make the structure meaningful for human and machines and allow automated design and orchestration of services, whereas BPEL does not [22].

The execution order of services is usually defined using a design tool (textual or graphical), which is then executed and monitored using an engine. For BPEL, Apache BPEL Designer and JBoss Tools BPEL Editor can be given as examples to design tools, whereas Oracle BPEL Process Manager, Apache ODE, IBM WebSphere Process Server, and Microsoft BizTalk Server can be listed as examples for execution and monitoring.

Using OWL-S increases the interoperability and enables automatic orchestration between the services, but it requires a deep knowledge in the domain. Hence, there are few editors available for OWL-S. Additionally, to create complex workflows, Protégé OWL-S Editor [23], which is a plug-in for Protégé, can be utilized. Nevertheless, the usage of this plug-in also requires advanced knowledge in the domain. All of these tools must be locally installed to be used. To convert Web services into Semantic Web services, a design tool and an execution engine are necessary.

In another approach, created in the context of the THESEUS funding program, a framework for the discovery, integration, processing, and fusion of Semantic Web services is described [24]. According to a user request, the framework identifies and assembles matching services for problem solving and creates a plan for the composition and execution order. The focus is on the matching of heterogeneous services and the fusion of all gathered information in real time. The harmonizing and mapping of knowledge is carried out based on ontologies.

The advantage of our approach is the continuous integration of services in an cloud-based infrastructure. The user is guided from the provided Web user interface to a graphical editor, where individual services could be integrated, experiments could be designed by orchestrating these services and stored as specific workflows that could also be executed within the framework in a further step. This execution of workflows could be initiated by the creator of the workflow or by another authorized user who is allowed to get the results. Within the developed infrastructure, specific services can be deployed and assigned to simple or complex workflows graphically. Each provided tool can be used and executed without detailed knowledge of the underlying Semantic Web technologies. All functionality of the cloud-based infrastructure platform can be accessed using a simple Web browser without installation of additional software applications.

III. Scenario

In the engineering domain, a conventional practice for quality assurance of the manufactured final product are comparison checks against the virtual designed product model. This accuracy check is performed by comparing two 3D models. First a scanning process creates and transfers accurate points, and in this way a virtual 3D model is created. The entire model consists of millions of 3D points, which must be matched and compared with the designed product model to find out the discrepancies by calculating the distances of points in both, the designed model and the virtual clone of the final product [25][26].

The manufacturer of these big turbine blades uses different tools to perform this comparison task and these supplementary tools generate additional license and training costs. The handling of different software solutions requires many hours of work. By using the workflow and service infrastructure and the distributed HPC solution described here, the comparison time is significantly reduced. These advantages allow the company to focus on quality measurement and also increase the capacity of the company for initiating new projects. Figure 4 shows a complete Kaplan turbine (a), one blade that is to be evaluated (b) and the scanned and virtualized 3D model with color-coded comparison results (c) [26]. The virtual model is created by an open-source tool for rendering and visualization [27].

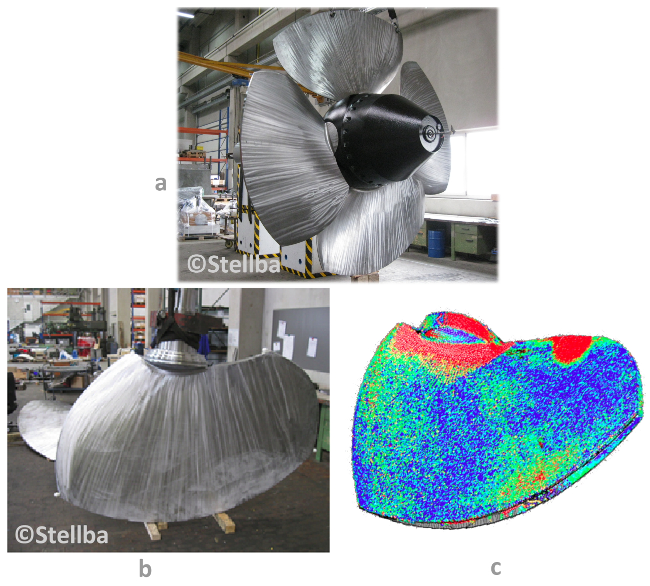


Figure 4. A complete manufactured Kaplan turbine (a), its single turbine blade (b) and the color-coded comparison of its scan and design (c).

The workflow described in this scenario consists of several orchestrated services. In the first step a service is included, which allows to load files as input for the experiment. In this scenario two files, a Computer-Aided Design (CAD) model and a laser scanned point file are chosen. With the help of another service that provides an user interface an initial coarse alignment of both models can be made manually. Finally, the automated comparison of the models is performed in the next service that uses the HPC cluster. This is helpful, in order to shorten the comparative process by increasing the computing power. The result of this calculation is displayed in a dedicated application, the distance viewer. The details on this scenario can be found in the work done by Holm et al. [4].

IV. Concept

This approach follows the idea of offering individual functionality by services via standardized interfaces. These can be easily integrated, offered and consumed via a cloud-based infrastructure platform. Unfortunately, conventional service descriptions do not provide enough self-description capabilities. Therefore, in the concept of this approach, a language has to be chosen, which allows semantic descriptions to add more contextual information to the services. There are a number of semantic technologies that provide machine and human readable descriptions, such as OWL-S, Web Service Modeling Ontology (WSMO), Web Service Modeling Language (WSML), and the Semantic Web Services Framework (SWSF) [28]. The most recently updated language to describe Semantic Web services, OWL-S, is used to provide descriptions that are machine readable and processable.

In this approach OWL-S satisfies the requirements and is chosen to describe the services and their orchestrations, describes further on as workflows. In addition to the description of the technical parameters, each service must also provide the information on the provided functionality, e.g., analyzing methods and calculation models such as the finite element method or the visualization and comparison of 3D-models.

The used ontology, which follows the OWL-S standard, is shown in Figure 3. Based on this upper-level ontology, a Service and Workflow Ontology (SWO) is created to store domain-specific information for interoperability of services and workflows. This SWO with its concepts, instances, and the relationships between them, is shown in Figure 5. The defined SWO makes it possible to differentiate the service categories from each other via *CFServiceType* class. It also introduces *User* and *UserGroup* for permission control, which is one requirement to separate users in multi-user environments. Based on the described *InputParameter* and *OutputParameter* classes suggestions for technically matching services can be made, during the process of the creation of an individual experiment.

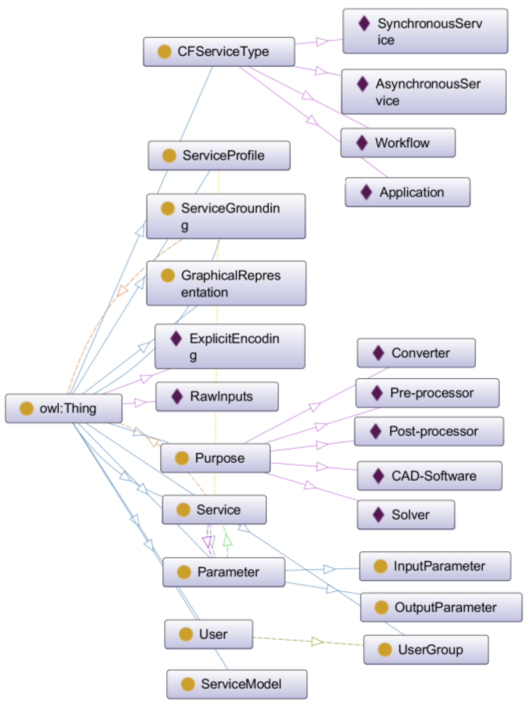


Figure 5. This ontology stores context-specific service and workflow information.

Due to the complexity of OWL-S and the necessary embedding into the application domain, all available technical Web service interfaces need to be described using standards, such as WSDL. The semantic service description is generated out of these WSDL descriptions. All necessary additional information is offered to the user by forms within the Web portal and transferred to the description. Based on this technical description with its functions, input, and output types, a Semantic Web service model, described in OWL-S, is generated and integrated into the Semantic Repository. This transformation and conversion is automatically performed by provided converter library, Semantic Web Service Creator, shown in Figure 6. As depicted, the creator retrieves the URL of the WSDL file, the service type and user provided semantic information for the concrete data types of service inputs and outputs in addition to the data types parsed from WSDL. In a next step, it converts the service into a specific instantiated Semantic Web service description in OWL-S that is stored in the Semantic Repository.

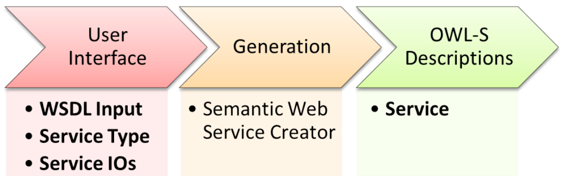


Figure 6. Generation of services in OWL-S.

In principle, according to the different design of the experiments, two service categories are differentiated. On the one hand, services that have a role to play in delivering data within a specified period of time, and on the other hand, those long-lasting services that do not matter that the data quickly land at the addressee. In this approach, we used these services types in three different application categories:

- Synchronous Service
- Asynchronous Service
- Asynchronous Web Application

These application categories are disjoint, i.e., each Web service can only be assigned to exactly one category and stored in the Semantic Repository. Standard Web services belong to the *Synchronous Service* category. Whenever a request is made, they must respond within 60 seconds, which is defined as default timeout limit in HTTP. Every service, which does not require an interaction with the user, is part of this category. An *Asynchronous Service* is a special category that the services within return information whenever the calling component checks the status. Unlike the previous category, asynchronous services can display feedback messages and these services can last days or even weeks to complete. A response to the calling component reports the status by telling either the service is completed or still ongoing. Lastly, the *Asynchronous Web Application* category contains Web services, which are similar to asynchronous services, but without a status check. This type of service is used to provide interactive user interfaces on Web pages within the Web portal. Since the completion of the tasks for this service type depend on user interaction, a trigger must be sent to the called for reporting.

Services provide functionality for special tasks, but complex tasks in the engineering domain usually consist of multiple steps. Therefore, one service is not sufficient and an orchestration of services is needed. The cloud-based infrastructure allows users to create experiments and orchestrate services individually, formalized by a workflow. Depending on the application domain, a workflow can have multiple definitions, but in the context of this paper, a workflow will be considered as a set and ordered list of chained up Web services provided with a service description in WSDL and with SOAP-based APIs to involve, deploy and perform each specific task with or without user interaction. According to the different requirements of the experiments to the editing and execution environment, four different constructs were defined in order to be able to model corresponding workflows, as depicted in Figure 7.

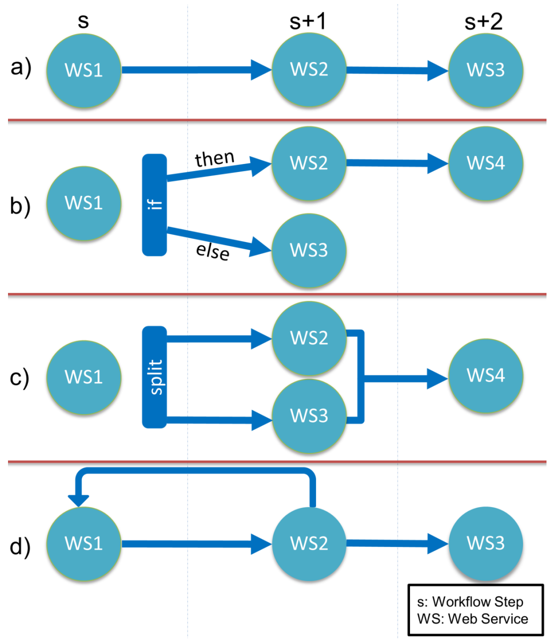


Figure 7. Possible workflow design and execution types.

A workflow of an experiment can of course be designed with a combination of these and consist of one or more of the listed types. The following workflow design and execution types are possible using the OWL-S:

(a) A linear, sequential execution of several Web services, which often fits for many situations. The services within the workflow will be executed one after another, one at each workflow step.
(b) Branching the workflow with *if-then-else*, which follows one of the two possible branches after evaluating the inserted condition. This is mostly useful for evaluation of a simulation and performing additional tasks in case the results are not satisfactory.
(c) Parallel execution of two branches simultaneously via *split*, in case their inputs and outputs do not depend on each other. This type of workflow is generally useful if a sequential execution of multiple services would take long time and a separate execution of independent services would reduce this.
(d) Looping back to a specific step using *repeat-until*, until a condition is fulfilled. This type is useful to iterate specific step(s) until the result is the acceptable value.

As shown in Figure 8, a user interacts with the Web portal of cloud-based infrastructure platform via standard Web browser. A part of the portal is the graphical user interface for the creation and editing process of workflows. Each workflow describes an experiment and consists of several orchestrated services. Within this concept, the workflow editing component stores these individual experiments in an storage solution with attached annotations, the Semantic Repository, when the editing task is finished. Furthermore, independent of the editing step, the user can initiate the execution of stored workflows of the respective experiments. The result of each conducted experiment is delivered to the Web portal by the execution component of the workflow management and execution component.

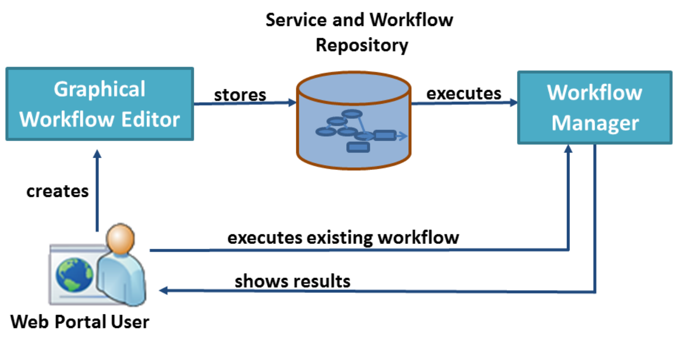

Service and Workflow Repository
Graphical Workflow Editor
stores
executes
Workflow Manager
creates
executes existing workflow
shows results
Web Portal User

Figure 8. User creates a workflow of an experiment and initiates the execution.

This suggested kind of service categorization and interaction make it possible to support users and their specific needs to complete their tasks with synchronous or asynchronous processes executed in the background. A service orchestration is performed by using a component for workflow editing to create a semantic workflow description. First, the workflows are created or edited and the services are arranged in the correct order, in which they have to be executed. In the process chain, the output of a service is passed to the input of the next service. This can be done by an supported graphical editing tool. The use of a tool has the advantage that the complexity remains hidden from the user. The user must not have a detailed knowledge of OWL-S to describe their workflow. The graphically sketched sequence of services is formalized in a workflow and stored in an XML-based meta-format that serves as input for the conversion into OWL-S. The automatic conversion of these inputs into workflows is also performed by the aforementioned Semantic Web Service Creator. Figure 9 shows the process of the generation of workflow descriptions for different experiments. Both processes in Figure 6 and Figure 9 use the same library for conversion. However, the inputs and outputs of the converter shown in bold text are different depending on the purpose. The former is for Semantic Web service creation whereas the latter is for workflow creation.

The service domain is structured by an upper model for the generic description and vocabulary of services and workflows in OWL-S. It defines how services must be described and specified, using annotations and technical descriptions. The knowledge of different application domains is represented by several domain ontologies that describe application functionalities in detail. The SWO provides properties to define several additional relations. The next section describes the special tooling to design individual workflows, where each step in the workflow

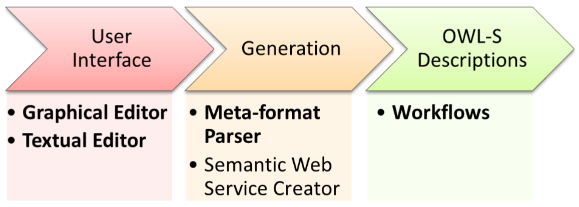


Figure 9. Generation of workflows in OWL-S.

must be assigned to a service.

V. Interaction of Core Components

The creation of an interoperable and flexible platform provided as IaaS requires an embodiment of several core components which are compatible with each other [4]. These core components are presented as Web services with their technical interface description in WSDL. The implemented core components to form the infrastructure and host all the functionality are:

- Semantic Repository for Services and Workflows
- Workflow Editor
- Semantic Web Service Creator
- Workflow Manager

Generic platform services are utilities which can be used for generic purposes such as loading data structures with different formats, e.g., Computer-Aided Engineering (CAE) and Computer-Aided Design (CAD) data. On the other hand, vendor-provided Web services introduce and wrap functionality for specific software components of different complexity levels. The services can encapsulate complex calculations and comparison operators or even provide the interface to third-party systems to perform complex calculations in HPC clusters. *Semantic Repository for Services and Workflows* component hosts the OWL-S descriptions of these services as well as workflows in triple store format. As a requirement by the project, a Web-based solution for easy integration of new services was necessary. Tools which works with OWL-S did not provide either a Web interface or were too complex for new users. Therefore, a wrapper and interface for easy conversion between WSDL and OWL-S is implemented. New services can be integrated with the help of a graphical user interface starting with a technical description of the service in WSDL. Using this description and additional information added to forms, the services are automatically converted into Semantic Web services, without requiring specific knowledge of used complex Semantic Web technologies, such as OWL and OWL-S. The service providers are able to deploy the WSDL description of their Web services via the *Workflow Editor (WFE)* component using the Web portal. This component assists in the integration of Web services in the Semantic Repository and creation of workflows. Unlike existing tools that available for different technologies as mentioned by Grolinger et. al. [28], the implemented WFE performs all tasks without installation of any software on the end-user side and targets all users with different knowledge in the domain. The *Semantic Web Service Creator* uses the absolute URL to the WSDL description of the service to generate the Semantic Web service descriptions in OWL-S. Then it stores and registers them in the Semantic Repository, including the inputs and outputs of the services. In another context, this component creates semantic workflow descriptions in OWL-S, based on the XML meta-format introduced by the WFE and stores the creator of the workflow along with their user group. A partial and incomplete example of this meta-format is given in Figure 10. The basic requirement for all saved services and workflows are unique names which are created automatically and a unique identifier of the user to set the ownership of a workflow. This identifier, a session token, is retrieved using an external identity service, OpenStack Keystone, therefore, no usernames or passwords are stored in the semantic repository [29]. The Semantic Repository is based on a central domain model, formalized as an OWL ontology, that describes the input and output types for the matchmaking process of the services. Finally, the *Workflow Manager (WFM)* component starts, manages, orchestrates, monitors workflows, and checks permissions of the users for the execution. This component is also responsible for passing the data between services.

```
<workflow URI="namespace/Workflow.owl#Name">
  <input ID="input1"
      URI="namespace/workflows/Workflow#extraParameters"
      x="-6" y="527"/>
  <input ID="input2" URI="namespace/workflows/Workflow#file"
      x="53" y="623"/>
  <input ID="input3"
      URI="namespace/workflows/Workflow#sessionToken"
      x="55" y="721"/>
  <output ID="output1"
      URI="namespace/workflows/Workflow#DistanceFile"
      x="1535" y="783"/>
  <services>
    <sequence>
      <service URI="namespace#CADFileService" x="387" y="43">
        <input URI="namespace#buttonText"/>
        <input URI="namespace#desc" value="Select"/>
        <input URI="namespace#filter" value="pts"/>
        <input ID="input3" URI="namespace#sessionToken"/>
        <input ID="input1" URI="namespace#extraParameters"/>
        <input URI="namespace#gssToken"/>
        <input URI="namespace#header_base64"/>
        <input URI="namespace#serviceID"/>
        <output ID="p1_output1" URI="namespace#fileSelected"/>
        <output ID="p1_output2" URI="namespace#status_base64"/>
      </service>
      <service>
          ...
      </service>
    </sequence>
  </services>
</workflow>
```

Figure 10. Workflow description in the XML meta format.

An overview of the interaction of the core components of the developed platform is given in Figure 11. The main user interface of the developed platform is a Web portal, and it translates user actions into core component specific requests, e.g., during workflow design, workflow and service execution, service monitoring, and result management. The communication between the Web portal and the components are performed using SOAP messages therefore these requests are converted into this format according to the API. With the graphical interface of the WFE, the user gets access to the services stored in the Semantic Repository. Here, services dedicated to experiments can be chained up to create dedicated workflows, such as for the comparison of 3D models. Each created workflow is also stored in the Semantic Repository and can be found easily by simple properties. Determined by the complexity, multiple workflows can be necessary to finalize and perform the tasks needed for the whole experiment. In this case, semantic descriptions can use workflows similar to a Semantic Web service as "sub-workflows". Similar to workflows, sub-workflows are also stored in semantic repositories. Both of them are comparable to Semantic Web services, and reusable. Moreover, their descriptions are updated without causing any fragmentation.

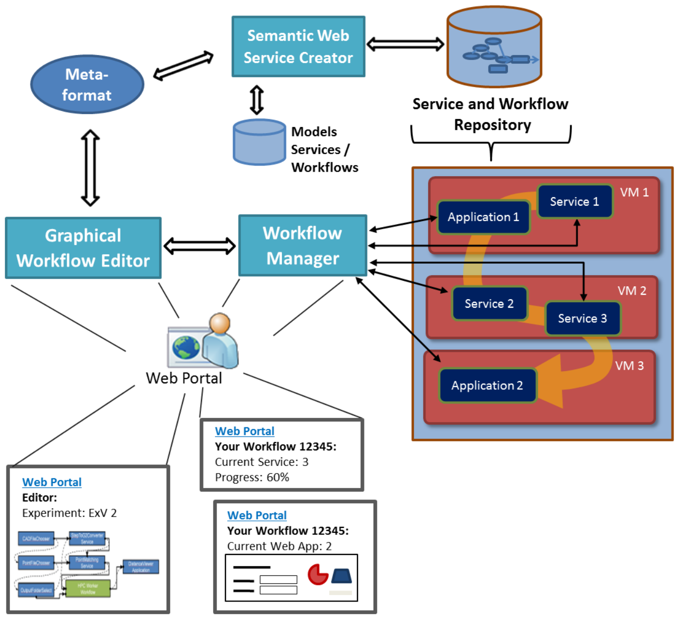


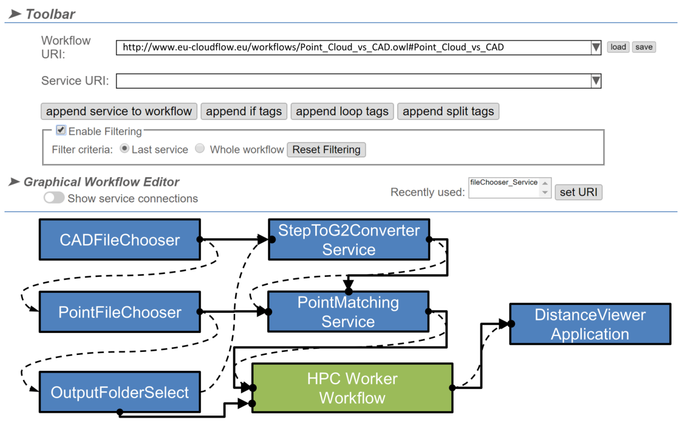
Figure 11. Overview of the interactions of the core components.

To store the experiments, the graphical contents of the related workflows are transferred into a XML-based meta-format. This format serves as input for the Semantic Web Service Creator that generates the workflow descriptions in OWL-S. The WFE is provided as a Web service which is described in WSDL. Additionally, all methods that available in its description are accessible through its graphical implementation. The graphical implementation of the WFE is currently developed using PHP and JavaScript, but can completely be written using another programming language. To allow that and prevent language lock-in, the workflows can also be defined using the defined meta-format. This format is also easy to understand and sometimes preferable by experts for fine-tunings. The purpose of graphical interface of the WFE is to provide a full-featured yet simple user interface to translate visual actions into this meta-format and then to prepare SOAP messages. These messages are later sent to the service implementation of the WFE.

The WFM component is used for the management and execution of individual predetermined workflows. In the execution task, the component processes the individual workflows and accordingly queries the listed services in the defined order. If the service execution is completed and the answer of a service is received, the next step in the workflow is activated. The results of respective services are unified and added to a single representation structure, which is passed at the end of all processing steps back to the UI of the Web portal.

The WFM can execute the workflows designed following any of the depicted types in Figure 7. For the cases *b* and *d*, the next step is determined by a logical expression. These logical expressions are defined during the workflow design. For user friendliness, these expressions are adapted and simplified, and can be one of *greaterThan*, *greaterOrEqual*, *equalTo*, *lessThan*, or *lessOrEqual*.

An example of a graphical workflow is shown in Figure 12. Using the toolbar, one can append the services into the workflow choosing a service to add and using *append service to workflow* button. They can also add snippets for branching (if-then-else), iteration (repeat-until), or parallel execution (split) using the corresponding buttons.


➤ Toolbar

Workflow URI: http://www.eu-cloudflow.eu/workflows/Point_Cloud_vs_CAD.owl#Point_Cloud_vs_CAD

Service URI:

[✓] Enable Filtering

Filter criteria: [● Last service] ○ Whole workflow

➤ Graphical Workflow Editor

Show service connections

Recently used: fileChooser_Service

Figure 12. The graphical workflow editor UI shows the scenario workflow.

The more the list of available services within the Semantic Repository grows, the more difficult the user finds it to choose the appropriate service for the workflow. To prevent this, the WFE introduces different filtering mechanisms. The service list has auto-complete functionality which allows searching as the user types. Although this helps to find the service that the user is looking for, it does not, however, provide the information about the compatibility of this service. To solve this, the WFE offers optional filtering method using the semantic information of the services. During service integration into the Semantic Repository, the users provide the acceptable types or file formats for the inputs and outputs of the service. During workflow design, when a service is appended into the workflow, the service list is updated containing only the compatible ones. This filtering has two modes which may suit for different situations. *Last service* mode considers only the last service appended into workflow and lists the services that are only compatible with this one. This is an aggressive mode which is useful if there are too many services registered in the Semantic Repository and only fully compatible services are desired. *Whole workflow* mode provides rather a moderate filter which also takes the previously appended services into consideration during filtering. To illustrate both modes, one can assume that there are several services denoted as S registered in the Semantic Repository R as:

$$\forall S \in R = \{A, B, C, D, E, F\} \tag{1}$$

where each S in R is compatible with the services written in their indices listed in a compatibility list L:

$$L = \{A_{B,C}, B_D, C_{D,E,F}, D_{A,B}, E_F, F_{B,D}\} \tag{2}$$

According to L, for example, $A_{B,C}$ is an integrated service that is compatible with services B and C. If $A_{B,C}$ is appended into the workflow as the first service, with the *Last service* mode enabled, the list will be updated to display only B_D and $C_{D,E,F}$. With this mode still on, when B_D is appended, the list will only display $D_{A,B}$ since this is the only compatible service with B_D. On the other hand, if *Whole workflow* mode is active and $A_{B,C}$ is appended, similarly, B_D and $C_{D,E,F}$ will be listed. When B_D is chosen from these two, the list will contain $D_{A,B}$ as well as B_D and $C_{D,E,F}$ since B_D and $C_{D,E,F}$ were compatible with $A_{B,C}$ which is an existing service inside the workflow. To increase the compatibility of the services, the input and output types of the services must be well defined during the service integration. However, it might be the case that a service with untested compatibility among other services is desired for the next step. In this case, to list all services regardless of their compatibilities, *Reset Filtering* button can be used to disregard the relations. Similarly, the filtering can also be disabled temporarily by unchecking *Enable Filtering* button.

In addition to this two filtering modes, a history of recently used services is also kept. This reduces the work of the user during workflow creation by listing all used services during the active session for quick access.

The execution order in WFE is represented by dashed arrows inside WFE and the blue blocks are the individual workflow steps. If a service is selected to be executed on HPC by the user in the editing process, an HPC sub-workflow is used. In the graphical editor interface this is marked as a green block. This means that the WFM initiates the service execution in a dedicated HPC server environment.

The structure of an HPC sub-workflow in turn consists of a sequence of three tasks:

1) *pre-processing task* to generate the command to be executed by HPC process,
2) *HPC command task*, which receives the command by user interface and gives feedback to the user, and
3) *post-processing task*, which converts the output from the HPC process into application specific output.

If the workflow is complex and consists of more services, the service connections can be shown or hidden using the *Show service connections* toggle.

For each workflow, the manager initiates execution procedures and tracks the progress individually. As the user, who created the workflow, and their group are stored in the semantic repository during workflow creation, it is possible to prepare a list of allowed users for the workflow execution. Therefore, before starting a workflow, the WFM checks whether the user or group has permission to run it to prevent unauthorized execution. It also provides a monitoring functionality, which allows users to leave the workflow anytime and return at later stage to continue where they left off. This maximizes the benefits of such a cloud-based platform, supporting access anytime and from any desktop or mobile device with Internet access. If the workflow does not need user input, the WFM is even able to complete it automatically and display its results to the user at a later time. The usability can be extended by including reusable utility services such as an e-mail service which notifies the user who ran the workflow at any step.

As explained in the previous sections, services of different vendors can be used that are implemented by different programming techniques and run within the cloud on different application servers. Nevertheless, during the lifetime of a workflow, the user does not need to know, where the services are stored and how the data is forwarded to the next service. The manager component retrieves the service descriptions and performs the tasks without user notification and the complexity of all associated services within a workflow remains hidden from the user. Using the cloud-based approach, it is also possible to include a service within a workflow which could execute the services in an HPC cluster that reduces computation times.

Based on their defined service types, synchronous or asynchronous, services are differentiated and executed by the execution engine of the WFM. The aforementioned service for the pre- and post-processing tasks are implemented as *Synchronous Services*, because they take just a few seconds to execute. However, HPC command task must be implemented as *Asynchronous Service*, since the duration of a Web service execution cannot be predicted. The status of execution is monitored and provided via a status method. This result is provided to the user as HTML feedback and displayed on the Web portal.

If a Web service provides an UI to interact, the service must be implemented as an *Asynchronous Web Application*. Services that belong to this category are implemented similar to *Asynchronous Services*, but contrarily do not need to deliver their status. This service type explicitly tells the WFM that the task is completed. After receiving this notification, the WFM performs the next step and gives feedback on the Web portal.

VI. Conclusion and future work

This paper explained the concept and realization of a flexible cloud-based infrastructure platform, which involves Semantic Web technologies and tailored tools for the creation, execution, and management of workflows and conducted services by graphical user interface. The realized platform allows a seamless integration and combination of engineering services, and a controlled execution and monitoring of used resources. It satisfies the requirements for the development and execution of user-driven experiments defined as workflows, without requiring detailed knowledge on High Performance Computing or other underlying technologies.

The platform offers a Web portal that can be accessed via standard Web browser, without the need of installing additional software. New services can be integrated and orchestrated to workflows with the help of graphical user interfaces. Starting with a technical description of the service in WSDL and using additional information added to forms, services are automatically converted into Semantic Web services, without requiring specific knowledge of used complex Semantic Web technologies, such as OWL and OWL-S. With another graphical user interface, the Workflow Editor, these integrated services can be orchestrated within the meaning of the experiment and stored as workflow descriptions in an platform-wide accessible Semantic Repository. The component assists the user by allowing search and filtering incompatible services during workflow design. Another core component of the created solution, the Workflow Manager, is used to execute, orchestrate, and monitor the created workflows of the experiments. The result of each conducted experiment is in turn delivered to the user via the Web portal of the cloud-based infrastructure platform.

Another advantage of this approach is the combination of the created platform with high-performance servers. Complex tasks can be outsourced to these servers and this results in enormous time savings and allows the experts to carry out more experiments with products, which was omitted due to the complexity and the required computing power until now. Of course, the possibility to conduct these experiments leads to an enormous increase in expert knowledge.

In future, the Workflow Editor will be able to perform automated dynamic workflow design. A dynamic workflow formalizes an orchestration of services, supported by an automated matchmaking process that provides adequate services ordered by their confidence values, which is only possible using Semantic Web technologies. Furthermore, the Workflow Editor will be able to insert converter services into the workflow automatically, just for adjustment of input and output types, e.g., convert units of measurement and file formats.

Acknowledgments

This research has received funding in part by the European Union's Horizon 2020 research and innovation program under grant agreement No 680448 (CAxMan). And this work is also based on preparatory work, which was funded in part by the 7th Framework Program of the European Union, project number 609100 (project CloudFlow). The responsibility for this publication lies with the authors.

References

[1] V. Gezer and S. Bergweiler, "Service and Workflow Engineering based on Semantic Web Technologies," in Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016), International Academy, Research, and Industry Association (IARIA). IARIA, 10 2016, pp. 152–157.

[2] T. Barton, "Cloud Computing," in E-Business mit Cloud Computing. Springer Fachmedien Wiesbaden, 2014, pp. 41–52.

[3] A. Ranabahu and A. Sheth, "Semantics Centric Solutions for Application and Data Portability in Cloud Computing," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 234–241.

[4] H. H. Holm, J. M. Hjelmervik, and V. Gezer, "CloudFlow - An Infrastructure for Engineering Workflows in the Cloud." in Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016), International Academy, Research, and Industry Association (IARIA). IARIA, 10 2016, pp. 158–165.

[5] R. Cyganiak, D. Wood, and M. Lanthaler, "Web Services Architecture," W3C Working Group Note, 2004, [retrieved: March 2017]. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," W3C, W3C Note, March 2001, [retrieved: March 2017]. [Online]. Available: http://www.w3.org/TR/wsdl

[7] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," W3C Recommendation, 2007, [retrieved: March 2017]. [Online]. Available: https://www.w3.org/TR/wsdl20/

[8] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," Scientific American, 2001, [retrieved: March 2017]. [Online]. Available: http://www.jeckle.de/files/tblSW.pdf

[9] G. Klyne and J. J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation, 2004, [retrieved: March 2017]. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[10] P. Hitzler, M. Krötzsch, and S. Rudolph, Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009.

[11] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax," W3C Recommendation, 2004, [retrieved: March 2017]. [Online]. Available: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[12] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "OWL Web Ontology Language Semantics and Abstract Syntax," Feb. 2004, [retrieved: March 2017]. [Online]. Available: http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[13] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler, "Semantic Web Architecture: Stack or Two Towers?" in Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, F. Fages and S. Soliman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 37–41.

[14] D. Martin et al., "OWL-S: Semantic Markup for Web Services," 2004, [retrieved: March 2017]. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[15] S. Bergweiler, "A Flexible Framework for Adaptive Knowledge Retrieval and Fusion for Kiosk Systems and Mobile Clients," in Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014), International Academy, Research, and Industry Association (IARIA). IARIA, 8 2014, pp. 164–171.

[16] M. Loskyll, I. Heck, J. Schlick, and M. Schwarz, "Context-based orchestration for control of resource-efficient manufacturing processes," Future Internet, vol. 4, no. 3, 2012, pp. 737–761.

[17] C. Casanave, "Designing a Semantic Repository - Integrating architectures for reuse and integration," 2007, [retrieved: March 2017]. [Online]. Available: https://www.w3.org/2007/06/eGov-dc/papers/SemanticRepository.pdf

[18] "Webster Database Definition," [retrieved: March 2017]. [Online]. Available: http://www.merriam-webster.com/dictionary/database

[19] Ontotext, "GraphDB - Semantic Repository," [retrieved: March 2017]. [Online]. Available: http://ontotext.com/knowledgehub/fundamentals/semantic-repository

[20] Sesame Framework Contributors, "Sesame Java Framework," [retrieved: March 2017]. [Online]. Available: http://archive.rdf4j.org/users/ch01.html

[21] "Web Services Business Process Execution Language Version 2.0," OASIS Web Services Business Process Execution Language (WSBPEL) Technical Commitee, 2007, [retrieved: March 2017]. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[22] S. Bansal, A. Bansal, G. Gupta, and M. B. Blake, "Generalized semantic Web service composition," Service Oriented Computing and Applications, vol. 10, no. 2, 2016, pp. 111–133.

[23] D. Elenius et al., "The OWL-S editor - a development tool for semantic web services," in ESWC, 2005, pp. 78–92.

[24] S. Bergweiler, "Interactive service composition and query," in Towards the Internet of Services: The Theseus Program. Springer Berlin Heidelberg, 2014, pp. 169–184.

[25] C. Stahl, E. Bellos, C. Altenhofen, and J. Hjelmervik, "Flexible Integration of Cloud-based Engineering Services using Semantic Technologies," in Industrial Technology (ICIT), 2015 IEEE International Conference on, 2015, pp. 1520–1525.

[26] Stellba, "Comparing CAD Models with 3D Scanned Manufactured Parts on the Cloud," [retrieved: March 2017]. [Online]. Available: {http://eu-cloudflow.eu/experiments/first-wave/experiment_6.html}

[27] C. Dyken et al., "A framework for OpenGL client-server rendering," in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference, 2012, pp. 729–734.

[28] K. Grolinger, M. A. M. Capretz, A. Cunha, and S. Tazi, "Integration of business process modeling and web services: a survey," Service Oriented Computing and Applications, vol. 8, no. 2, 2014, pp. 105–128. [Online]. Available: http://dx.doi.org/10.1007/s11761-013-0138-2

[29] H. H. Holm, J. M. Hjelmervik, and V. Gezer, "CloudFlow - an infrastructure for engineering workflows in the cloud," in UBICOMM 2016: The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies. IARIA, October 2016, pp. 158–165.