The CloudFlow Infrastructure for Multi-Vendor Engineering Workflows: Concept and Validation

Håvard Heitlo Holm*, Volkan Gezer†, Setia Hermawati‡, Christian Altenhofen§, and Jon M. Hjelmervik*
*Heterogeneous Computing Group
SINTEF Digital
Oslo, Norway
Emails: havard.heitlo.holm@sintef.no
and jon.m.hjelmervik@sintef.no
†Innovative Factory Systems (IFS)
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
Email: volkan.gezer@dfki.de
‡Human Factors Research Group,
University of Nottingham,
Nottingham, United Kingdom,
Email: setia.hermawati@nottingham.ac.uk
§Interactive Engineering Technologies (IET),
Fraunhofer Institute for Computer Graphics Research IGD,
Darmstadt, Germany,
Email: christian.altenhofen@igd.fraunhofer.de

***Abstract*—In this paper, we present the CloudFlow Infrastructure, which aims to provide an independent platform for engineering workflows, leveraging both cloud and high performance computing. Each workflow can combine software from different vendors, promoting interoperability through open standards, and easy access to data and compute resources. Here, we focus on the technological uniqueness of the infrastructure, and how end users within the manufacturing industries have validated it for real world applications. We also describe how high performance computing and remote desktop applications easily are integrated in cloud-enabled workflows. The infrastructure provides an easy-to-use platform for software providers to offer their software in the cloud and get access to a new distribution channel. At the same time, small businesses get pay-as-you-go access to advanced multi-vendor software solutions that will improve their products.**

***Keywords*–Workflows; Cloud computing; HPC; Semantic descriptions; One-stop-shop.**

I. Introduction

Cloud computing is currently becoming a natural part of the daily life, both for professionals and consumers. Users are already expecting to have access to all their data independent of which computer they are using, and they will soon expect the same behavior for advanced engineering tools. An ideal engineering workflow consists of software from different vendors, operating on the same data. A cloud platform that provides efficient and user-friendly workflows by combining different tools, is therefore needed to meet these expectations.

In manufacturing industries, different software suites are used across the lifetime of their products, including design, numerical analysis, quality assurance and maintenance. Furthermore, engineering software solutions are often computationally demanding and designed for parallel execution in a high performance computing (HPC) environment. Small and medium-sized enterprises (SMEs) in this market often find it too expensive to install the different solutions locally, due to hardware costs, installation overhead and license costs. This may cause loss in quality in their products due to insufficient analysis, and increased time to market and overly expensive design phases due to inefficient work procedures. For such companies, having access to a cloud solution that spans over different clouds and software providers, all integrated in tailored workflows, will not only save time and cost, but also improve their final products.

In this paper, we present the CloudFlow Infrastructure [1], [2], which is a cloud-based solution where users can execute workflows consisting of software from one or multiple vendors, providing ubiquitous access to compute resources, software and data. Workflow orchestration is made available through a workflow design tool, using semantic information associated to available software and its input and output data. Workflow execution is then automatically managed and monitored by a workflow execution tool, which acts upon these semantic descriptions. Data is passed automatically between the stages of the workflows, providing a seamless user experience. Through the use of open standards and cloud interfaces, interoperability between software from different vendors is obtained. The workflow is ignorant of the underlying operating systems, and whether simulations are executed in a cloud or HPC environment. The CloudFlow Infrastructure is not only attractive for end users, but also allows software vendors to reach new customers through a pay-per-use distribution channel for their existing or new software solutions.

In contrast to other approaches, the aim here is to integrate existing software solutions into one common platform, combining them to work together as multi-vendor workflows. Furthermore, the proposed solution supports installation in

private clouds, as well as access to multiple cloud and HPC providers through one common web portal. To facilitate a broad selection of existing software solutions, the CloudFlow Infrastructure target cloud providers offering Infrastructure as a Service (IaaS), where software providers can install their own operating systems and fully control the virtual machines.

The private cloud option allows for deployment in environments without network communication to the outside world. This also opens up for companies that for security reasons require that data is stored within their computing facilities, can take advantage of the user-friendly multi-vendor workflows.

The CloudFlow Infrastructure has been continuously tested and evaluated by SME end-users in the manufacturing industries. The evaluation results have been used to improve the usability of the infrastructure, as well as the workflows developed by the software vendors. The validation has shown that the infrastructure is particularly useful for *engineering apps*. An engineering app is a design centric workflow that is applied to concrete artefacts such as pumps, structures, wings, etc., and is easily employed by end-users without domain knowledge nor experience with dedicated software tools. These apps makes it feasible for the involved manufacturing end-users to use more advanced technology than today, reducing design cost as well as time-to-marked. The involved software vendors and consultancy companies are given an opportunity to gain economic benefit through a partnership with an SME start-up that will offer engineering apps as a service. The technology behind the CloudFlow Infrastructure is not restricted to manufacturing industries, and can be applied to any other computationally intensive domain.

This paper is an extension and improvement of [1] with respect to end user validation of the infrastructure and its deployed software solutions; support for more advanced flow control in the workflows; and workflow integrated remote access to desktop applications executing in the cloud environment.

The paper is organized as follows: An overview of related work is given in Section II. The CloudFlow Infrastructure is then presented in Section III, where the focus is on the aspects and infrastructure components related to workflow orchestration and execution, resource monitoring, authentication, data storage, utilization of HPC clusters, and remote access to desktop applications. Section IV describes the methods for end user validation and discusses the validation results. A detailed example of a workflow running in the CloudFlow Infrastructure is given in Section V, before Section VI gives some concluding remarks and discusses the future of the infrastructure.

II. Related Work

Several providers currently deliver cloud-based engineering and computing solutions. One dedicated software vendor delivering such a solution is SimScale [3], offering simulators for computational fluid dynamics, finite element analysis and thermodynamics in the cloud. Based on these simulation tools and web-based visual pre- and post-processing, SimScale targets end users only. Combining their cloud solution with software developed by other vendors is therefore not straightforward.

The cloudSME project [4], [5] combines a business model targeting both end users and software vendors. Software vendors are offered a Platform as a Service (PaaS) solution, where they offer Software as a Service for their existing and new end users. This approach also makes it possible for end users to combine software from different software vendors to perform more complex engineering tasks. CloudSME does however not use any semantic information to orchestrate or combine the different software, and it lacks the use of HPC.

The Fortissimo and Fortissimo2 projects [6], [7] also offer a platform that supports business models for both end users and software vendors. They do however not target the cloud aspect, and mainly offer a platform where independent software vendors provide HPC simulations to end users. Similar to CloudSME, Fortissimo does not orchestrate combinations of different software based on semantic information.

There are several initiatives to simplify the process of deploying software in the Cloud. Ferry et al. [8] propose a modelling approach, where the cloud deployment is described by a vendor independent language CloudML. Deployment models are implemented in this language and can include descriptions of virtual machines, definitions of network communication, and instructions for service deployments. In addition to simplifying deployment of interconnected software, the language aims at helping their users avoid vendor lock-in.

Multiple approaches exist to gain remote access to software deployed in the Cloud. The platform-independent Virtual Network Computing (VNC) [9] and Microsoft's Remote Desktop Protocol (RDP) [10] are widely used to share applications across a local network or the internet. Both transmit images of the remotely running software or the remote machine's entire desktop, and receive the user's mouse and keyboard input for interaction. Network bandwidth and latency are crucial factors for a good user experience when using these technologies. Other techniques for remote visualization are, e.g., the Tinia framework [11] for interactive 3D data, or the Rixels approach for visualizing simulation results [12].

Stahl et al. [2] proposed the initial work and the main concepts of the CloudFlow Infrastructure. Among the newly introduced concepts are a unified way to access HPC resources, functionality to use external cloud providers, resource monitoring, and a graphical tool to define workflows.

Semantic Markup for Web Services (OWL-S) and *Business Process Execution Language* (BPEL) are two technologies that allow web service execution as processes. BPEL is a language for executing business processes with web services, as stated by Grolinger [13]. According to its specifications, BPEL executes web services defined using Web Service Description Language (WSDL). It supports orchestration of actions within such services, by structuring them as sequences and supporting branches and loops. The structure is described using a syntax based on Extensible Markup Language (XML). OWL-S is a markup that is built on top of Web Ontology Language (OWL) and describes web services semantically introducing an XML-based syntax. It also supports orchestration and due to semantic technologies, structuring the sequences using OWL-S is both machine and human understandable. OWL-S and WSDL are usually used to describe services based on the Simple Object Access Protocol (SOAP) specification. It allows web services to send requests in a predefined structure encoded in a XML format.

BPEL is similar to OWL-S in terms of orchestration and XML-based syntax, but it lacks utilizing semantic technologies. Therefore, making web services machine-understandable and automating them without user interaction is a non-trivial task using BPEL [14].

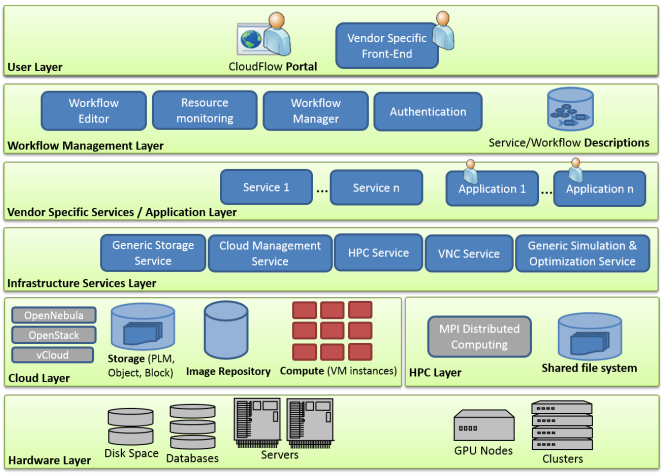

Figure 1. Simplified diagram of the system layers with their main components.

The process execution is usually performed by designing an execution order and monitoring it with an execution engine. There are several execution engines designed for this purpose. Execution engines, or managers, introduce an editor or a syntax to specify the order and then track the progress. Depending on the implementations, they can also provide user interface.

One of the available execution engines for BPEL is Process Manager by Oracle. It provides a graphical interface to manage cross-application business processes in a service oriented architecture (SOA) [15]. It also allows designing workflow steps and connecting external systems into the workflow. However, lack of semantic technologies inside BPEL prevents automation of these design steps. Involving semantic technologies will therefore increase the productivity by reducing the time needed to design the task steps, and is hence quite important.

To achieve the goal of CloudFlow, a manager that can facilitate semantic technologies, integrate web services from different providers and locations, and provide automation during the design and execution phase was necessary.

III. Architecture Overview

This section presents key components and concepts of the CloudFlow Infrastructure. The infrastructure can be described as a layered architecture, as shown in Figure 1. The layers represent different abstraction levels, and communication between layers is most often initiated in a downwards or sideways direction. In general, each layer consists of several loosely coupled components, where communication between the components is done through web services.

The *User Layer* is the user interface towards the CloudFlow Infrastructure, and is what the end user sees. This is typically the CloudFlow Portal, but can in principle be any application (e.g., web, mobile, desktop, etc.) that communicates with the components in the *Workflow Management Layer*. The workflow layer consists of components that are strictly needed for running any workflows in the infrastructure, and these components handle workflow management (Section III-A), resource monitoring (Section III-B), and authentication (Section III-C). The *Vendor Specific Services/Application Layer* contains all services that are executed as part of a workflow. Software provided and integrated into CloudFlow by independent software vendors belongs to this layer, and these services can only be accessed through the Workflow Manager. Components in the *Infrastructure Layer* are generic services that expose central functionality in the infrastructure. Some of the components can be used by services in the vendor specific service/application layer to access resources, such as cloud storage through the Generic Storage Services described in Section III-D. Other services can be used as individual workflow steps, such as the HPC and VNC Services described in Section III-E and Section III-F, respectively. All services mentioned above are deployed in the *Cloud Layer*, and the *HPC Layer* is available for executing computationally intensive applications. These layers yet again rely on the hardware found in the *Hardware Layer*.

A. Workflow Management

A workflow is an orchestrated and repeatable pattern of several activities enabled by the systematic organization of resources into processes that transform materials, provide services, or process information. Workflows may be as trivial as browsing a file structure and visualizing a Computer-Aided Design (CAD) model, or they can be more complex, including describing the full set of operations used to design, analyse and prepare a product for manufacturing. Semantic technologies, such as OWL-S, make it possible to design and automatically execute workflows.

As described in Section I, one of the main goals of CloudFlow is to host software from different software providers and chain appropriate parts of them to perform end user tasks in workflows within one common platform. In the following, we will describe how web services are integrated into the CloudFlow Infrastructure, and how workflows are designed and executed.

1) Services: A set of complementary reusable functionalities that are provided by a software for different purposes is called ”service.” More particularly, a web service is a software system designed to support interoperable machine-to-machine interaction over a network [16]. A web service invocation consists of a single request/response pair and is expected to execute in a short time.

The CloudFlow Infrastructure defines Application Programming Interfaces (APIs) that services have to follow in order to be integrated into the infrastructure and used within workflows. The services have to be exposed through a SOAP interface and defined by WSDL files. The simplest web services that follow those requirements are called *synchronous services*, and are used when their operations only take up to a few seconds to complete. In contrast, *asynchronous services* do not have any restrictions when it comes to execution times, and are suitable for long-running software executions without user interaction. For this kind of services, progress information is expected to be passed to the user through their service interface during the execution. Common for these two service types are that they represent operations that take predefined input parameters and generates output parameters without user interaction. Software designed for user interaction are made available as *applications*. Examples of applications are 3D CAD visualization software, web forms where users provide input parameters, and web interfaces to navigate in the cloud storage. Throughout this paper, the term *CloudFlow service* denotes services and applications compatible with the CloudFlow API.

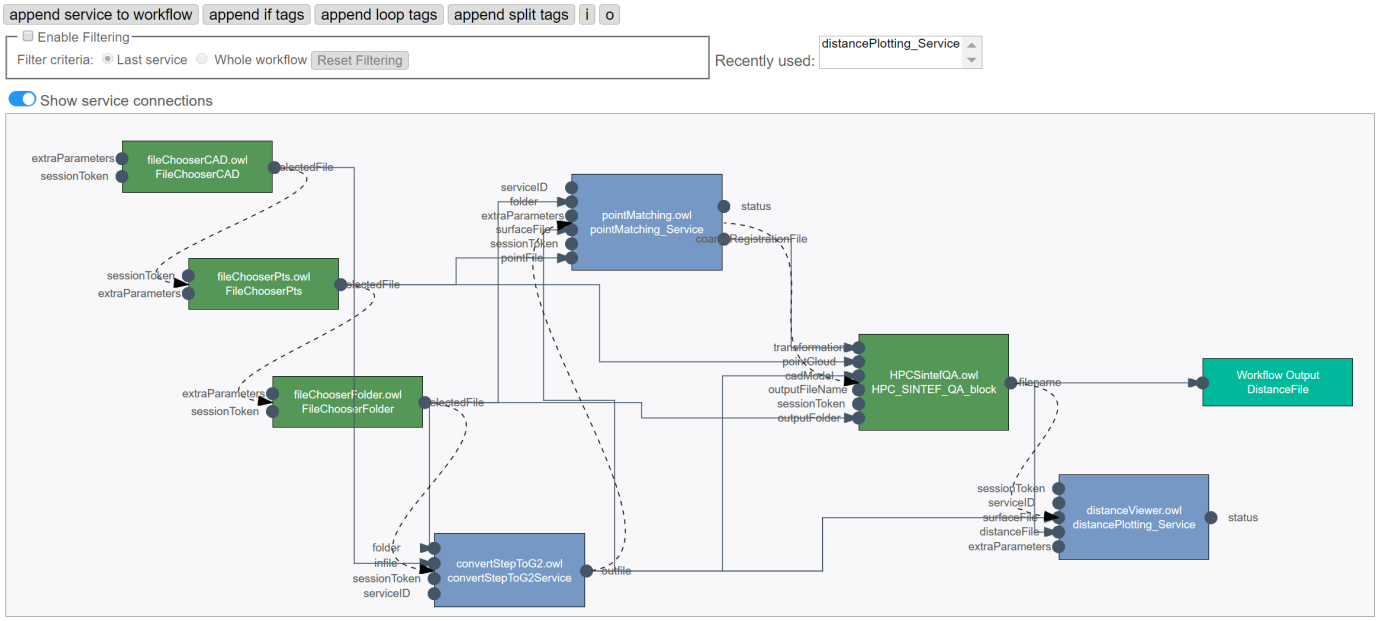

Figure 2. The graphical user interface of Workflow Editor displays buttons to append services and to add code snippets for conditional branches, loops, parallel service execution (split), etc., as well as options for service filtering. It also lets users add inputs and outputs to the workflow with appropriate buttons shown as I and O. The workflow is shown as a directed graph, where blue boxes represent single CloudFlow services, and dark green boxes are sub-workflows. The execution order is represented by the dotted arrows, while data flow is visualized by solid lines, which connect output parameters from services to be input parameters of others.

2) Workflow Definition: In order to transform a web service into a CloudFlow service, and then to create workflows from a chain of CloudFlow services, the web services need to be integrated into the CloudFlow Infrastructure. This integration is done using a tool named *Workflow Editor*.

Workflow Editor is a workflow modelling tool provided through both a graphical and textual interface [17] within the CloudFlow platform. It is based on XML, SOAP, and WSDL standards. In order to integrate web services into the CloudFlow Infrastructure, service providers submit the WSDL endpoints into a web form in Workflow Editor, and semantic descriptions are created, describing the services themselves and their input and output parameters. This information is then added to a semantic database, and the services can finally be used within workflow.

Workflows are created and edited through the textual and graphical editors of Workflow Editor. The data flow between services is defined by connecting outputs of services with inputs of others, using drag and drop functionality. The execution order is represented using dotted arrows and the data flow is visualized using solid lines, as shown in Figure 2. Based on the semantic description of a service, it is possible to find other services whose input parameters are semantically compatible with its output parameters. Semi-automatic orchestration of workflows is made available by letting the system suggest such compatible services to workflow designers.

All content in the graphical editor is also shown in an XML-based meta-formatted textual editor. The XML-based representation is sent to the Workflow Editor back-end, and contains all information required to save a workflow. This format is also sometimes preferred by experienced users. The textual and graphical editors are synchronized, so that each change made on a workflow in one of them is immediately reflected in the other.

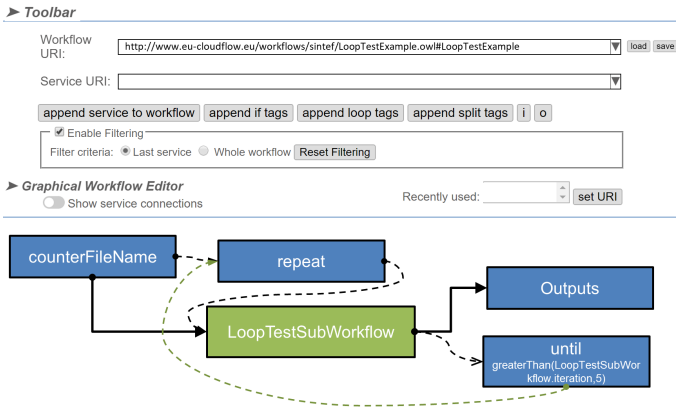

Figure 3. A simple example workflow for loop usage. The condition is modified by double clicking on the *until* block.

In earlier versions of the CloudFlow Infrastructure, all workflows were executed in a linear and sequential order [1]. However, it is sometimes desirable to design more complex workflows, where certain services are repeated, and different execution paths are triggered based on previous states. As non-sequential workflow execution was required by end users and software providers, support for conditional branches and loops have been added in the infrastructure. These functionalities are represented in the workflow design by the *If-Then-Else* and *Repeat-Until* OWL-S statements, respectively. In Workflow Editor, these expressions are adapted for simplification and compatibility with both the graphical and textual editor. The control of these expressions are handled via logical conditions such as *greaterThan*, *greaterOrEqual*, *equalTo*, etc., and they

can be set by double clicking on the *if* or *repeat* blocks in Workflow Editor. Figure 3 shows a very simple workflow consisting of a loop which iteratively execute a sub-workflow.

In some cases, workflows contain services whose parameters do not depend on each other, allowing them to be executed in parallel. It is possible to design and execute workflows containing parallel execution of different services, by using the *split* statement of OWL-S. A current limitation, however, is that the user can only interact with one of the parallel branches, meaning that the parallel section can only have one branch which requires user interaction in the form of applications.

Even though each CloudFlow service typically represents an individual operation with dedicated input and output parameters, some services naturally belong together. Instead of having to connect the same sequence of services repeatedly for multiple different workflows, such services can be modelled as smaller workflows of their own, called *sub-workflows*. Sub-workflows can be added as a single component into any other workflows, similar to a regular CloudFlow service. The changes made within a sub-workflow are applied to all workflows using it, reducing time and effort for the workflow designer through avoiding a repetitive task.

3) Workflow Manager: The semantic descriptions created by Workflow Editor contain meta-data and describe how the data is bound. The component *Workflow Manager* is an execution tool acting on the semantic descriptions. It executes and monitors all services in a workflow providing the input parameters, as defined in Workflow Editor, either as constant values or outputs from previous services. The status of each asynchronous service is checked at regular intervals in order to determine if it is finished, and to present the service's status to the user. The Workflow Manager itself consists of a back-end, which is working in a Java Virtual Machine, and a web service front-end, with which the Portal communicates. A simplified communication diagram of the components in the Workflow Management Layer is shown in Figure 4.

The main access point for starting, monitoring and interacting with workflows, is through the CloudFlow Portal. The workflow execution, however, is independent from the Portal, and non-interactive CloudFlow services are automatically executed by the Workflow Manager back-end, even if the user leaves the client that initiated the workflow. If a workflow reaches an application, Workflow Manager waits for user interaction before continuing to the next service. Since workflow execution is independent from the Portal, users can also re-login on any device and still have access to all running workflows, continuing from their current stage. Workflow results are stored in a MySQL database by the Workflow Manager back-end, and are accessible for the users at any time.

In order to have access to a workflow, the end user needs to have valid licenses for all services within that workflow. Before every workflow execution, license validation is done by Workflow Manager towards the billing services. Workflow Manager also tracks execution times by utilizing resource monitoring components.

B. Resource Monitoring and Billing

To facilitate that CloudFlow becomes a one-stop-shop where software vendors integrate and offer their software for new customers, functionality to monitor the resource usage by each workflow and service is needed. Based on different

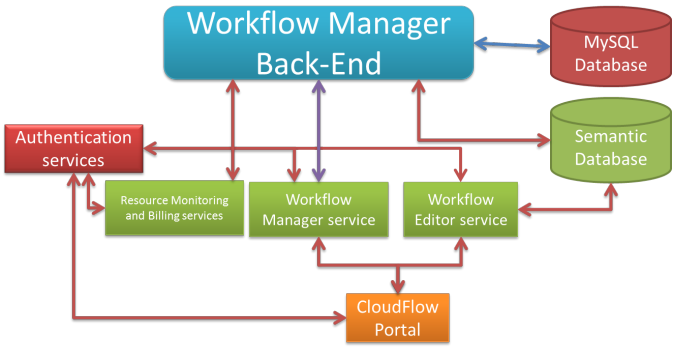

Figure 4. Simplified interaction between the components in the Workflow Management Layer in the CloudFlow Infrastructure.

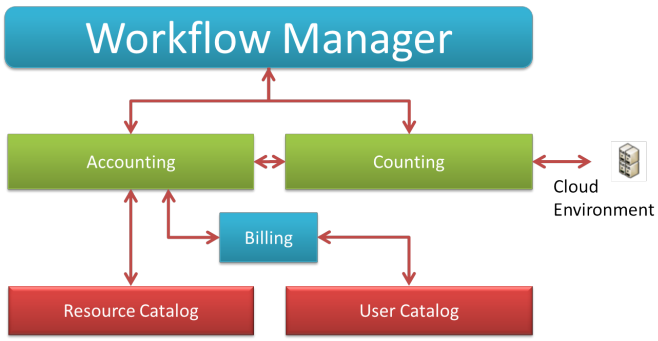

Figure 5. Communication between components related to billing and resource monitoring inside the CloudFlow Infrastructure. All components make additional calls to the Authentication services, which are not depicted.

requirements, the software vendors are able to use different business models, such as offering their software as pay-per-use, or for a fixed monthly or annual fee. For computationally intensive software requiring exclusive access to hardware resources, it is also natural to charge the end users based on the number of central processing unit (CPU) hours spent.

Each service within CloudFlow belongs to a *software package*. All license costs related to the software package are described in the *Resource Catalog* component, as either a time based license requirement, runtime cost, or a combination of both. The Resource Catalog also holds information about hardware costs and the vendors who will get paid from the revenue. Figure 5 illustrates how the different components are connected.

As a workflow is executed, the resources consumed through this workflow is gathered, and the related cost for running the workflow is calculated. Collection of this information and cost calculation is performed within the resource monitoring and billing components of the CloudFlow Infrastructure. When an end user starts a workflow, Workflow Manager lists all software packages within the workflow and checks with the *Accounting* service whether the user has the required licenses to run them. Later, if the user is allowed to execute the workflow, the *Counting* service is used to track the execution times delivered by Workflow Manager. This service passes these data back to the Accounting service to calculate the bill for CPU usage, as well as the software costs within the workflow.

The Resource Catalog holds a list of software and hard-

ware/data centers, whereas the *User Catalog* keeps a list of organizations or users. The calculations and usage information at the end of each workflow are gathered by the *Billing* component. This component issues bills to the end users and is the only component which users interact with in order to get their usage and cost reports.

C. Authentication and Multi-Cloud

Several of the CloudFlow Infrastructure components need to be available from outside the infrastructure itself. This includes, but is not limited to, design and execution of workflows, interaction with the cloud storage, and viewing how much resources a user has spent. Since only registered users should be allowed to issue such requests, and since users should only have access to their own files and resource usage information, all web services within the CloudFlow Infrastructure need an authentication parameter identifying the user. For this, a token-based authentication system is used. Users obtain a token at login which represents them throughout the session, and which contains their appropriate permissions.

As a security measure, tokens have limited life spans, meaning that requests containing old tokens will be unsuccessful. However, workflows (and perhaps especially within manufacturing industries) can consist of services lasting longer than any lifespan given to tokens. Since an expired token cannot be used for, e.g., uploading results to the cloud storage, CloudFlow needs an authentication scheme that invalidates tokens after a certain time while allowing workflows of arbitrary lengths to have access to all required infrastructure components.

In order to support these requirements, the *Authentication services* are introduced to CloudFlow. These services build on top of OpenStack's Keystone component [18], and extend it with functionality to handle the challenge related to long lasting workflows. In addition, vendor lock-in towards OpenStack is avoided through these services. Changing the communication with Keystone, or replacing Keystone itself, will require changes in the implementation of the Authentication services only, while its API, and all components relying on the authentication, are kept unchanged.

The problem consisting of tokens expiring during workflow executions is solved by issuing and storing special *workflow tokens* in the Authentication services. Each time a workflow is started, such a token is created by combining the regular token with the ID of the workflow execution. This workflow token is stored in a database, along with other relevant information, and is passed to all services within the workflow. During validation, the regular token is checked first, but if it has expired, the workflow token is checked with the database. As long as the combination of the regular token and the execution ID is found and recognized by the database, the token is still marked as valid. A new regular and valid token, holding the same permissions as the original token, can then be generated based on the database entry. When the workflow later is finished or aborted, the special workflow token is deleted from the database, and thus invalidated.

As CloudFlow is not tied to any one cloud, it is possible to use multiple clouds for hosting CloudFlow services. One reason for doing this might be that customers are physically too far away from the main CloudFlow cloud, making a local cloud more attractive in terms of network costs and delays. Other reasons might be that alternative clouds might be cheaper, or equipped with hardware not available in the CloudFlow cloud, for example by offering more powerful processing resources.

The main challenge related to such solutions is authentication across the different clouds. The external clouds have their own authentication methods, and they are not necessarily compatible with those used in CloudFlow. Services that are written for multi-cloud settings should therefore be implemented with an additional external token parameter. The semantic information can then describe which cloud the external token should be authenticated against, and external authentication services can be added to such workflows. Such a service will provide a web form where the user can login to the external cloud, and where the external token is passed to the next steps in the workflow. The external token can also be stored in a cookie in the browser, so that if a valid token is already present, this token will be used and the users are spared from typing their username and password more often than necessary.

D. Cloud Storage

In order to follow the loosely coupled layered architecture design of the CloudFlow Infrastructure (see Figure 1), the interaction with the cloud storage is designed to be vendor independent. CloudFlow supports various storage solutions and different cloud storage solutions have different APIs. Therefore, a set of services exposing a unified API for all storage solutions available in CloudFlow is required. Further, in order to avoid unnecessary network cost and to avoid potential security issues, files need to be transferred directly between the cloud storage and the client, and not via an intermediate server. To support these requirements, the *Generic Storage Services* (GSS) have been developed.

The GSS exposes an API consisting of both SOAP and REpresentational State Transfer (REST) web services, and offers functionality for interacting with all cloud storage solutions available in CloudFlow. In contrast to SOAP, RESTful web services come with a smaller overhead and are better suited for transferring large amounts of data. All file transfers are therefore done through RESTful services. Each available storage solution is added as a back-end to GSS, and SOAP services provide information in the form of a pre-defined recipe, on how to use the native REST interfaces. The client then follows this recipe to transfer files directly to and from the cloud storage, with no additional overhead. Through this design, GSS acts like a lookup service providing information on how to make requests toward the different back-ends, where each back-end is treated as an object storage. Beside transferring files, other functionality such as listing folder content, checking existence of files, creating folders, etc., are made directly through the SOAP API.

Unique references to files are obtained by assigning file IDs to them. A file ID includes a prefix indicating which storage back-end the files belongs to, and the location of the file within the native storage. The file ID and a valid authentication token is sufficient for downloading any file. As long as all CloudFlow services are implemented using this API, interoperability and vendor independent file access is obtained within CloudFlow workflows. Further, any cloud storage solution with a RESTful API can be made available in CloudFlow by adding an additional back-end in GSS. Existing services can then immediately use the new cloud storage solution without making any changes to their implementation. A cloud storage can also use external authentication solutions,

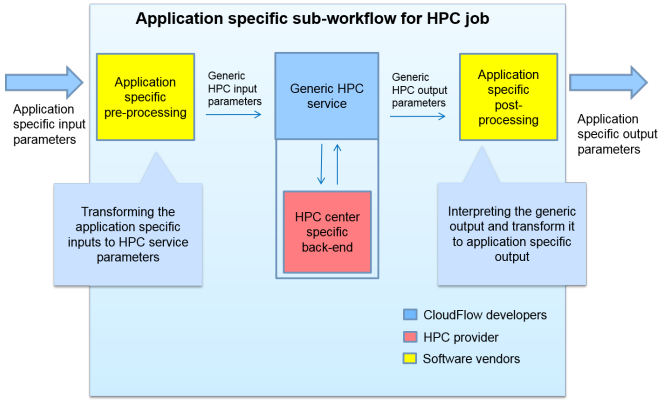

Figure 6. The encapsulation of a HPC job within an application specific HPC sub-workflow.

even though it is not a requirement when the authentication token used towards the cloud storage is a valid CloudFlow token.

A web-based file browser application is available for all workflows. It is configurable through Workflow Editor to tailor its behavior for each workflow, and has a user-friendly interface with a context dependent right-click menu. Here, end users can upload and download files between their computers and the cloud.

Since the CloudFlow services have SOAP interfaces, their parameters should consist of short messages rather than entire files. Because of this, the file browser gives the file ID of the chosen files as output instead of the content of it. This rule illustrate best practice and applies to all CloudFlow services.

Currently, four cloud storage solutions are made available in the CloudFlow Infrastructure through GSS. There are two OpenStack Swift installations (one internal and one external), a product lifecycle management (PLM) system, and a native storage at one of CloudFlow's HPC providers.

E. HPC Access

Computationally intensive tasks that benefit from running on HPC clusters are common in many engineering workflows. To support this, the CloudFlow Infrastructure facilitates seamless and secure integration of job scheduling, making it easy for software vendors to offer their applications on an HPC cluster as part of a CloudFlow workflow. This is supported through the design and concept of *HPC application sub-workflows*, with the generic *HPC service* as the central component.

An HPC application sub-workflow is built from three CloudFlow services as shown in Figure 6; an application specific pre-processing service, the generic HPC service, and an application specific post-processing service. The HPC service is designed to be generic with respect to both the application and the type of job scheduling system used by the HPC provider. Through this design, the HPC providers can make changes to their queuing systems, or CloudFlow can be expanded to more HPC centers, without requiring the software vendors to make changes to their services or workflows.

In order to make the service independent from details specific to the HPC center, the HPC service communicates internally with an HPC back-end through a pre-defined API. Since user credentials defined in CloudFlow are not necessarily compatible with the user definitions at the HPC center, the back-end may perform a mapping between the two sets of user definitions through a method seen fit by the HPC provider. The two existing HPC providers within CloudFlow currently use different approaches to solve this challenge:

- The HPC provider assigns a one-to-one mapping between the two types of users, providing each unique CloudFlow user with a dedicated user in their HPC infrastructure.
- The HPC provider has a set up pool of HPC users reserved for CloudFlow. Each execution of the HPC service is then assigned an arbitrary available user from the pool, and that user is then reserved until the execution is completed. To avoid that users can access other users' data, the home directory of each such HPC user is deleted between each job.

The first approach is particularly suitable to private cloud installations of CloudFlow, where the same system administrators control both the cloud and compute cluster environments.

The input and output parameters for the HPC service are designed to be highly generic, and should support the vast majority of applications that will be run on the compute cluster. The most important input parameter is the *HPC command lines*, which is the set of command lines to be executed through the HPC job. This typically includes loading required modules in the HPC system, downloading input files from the cloud storage, executing the application with the appropriate arguments, and uploading result files back to the cloud storage. Since the set of command lines is different for every HPC application, this parameter is expected to be created in an application specific pre-processing service. The idea is that this service has the same input parameters as the application, and generates a string as output that can be connected with the input parameter of the generic HPC service. Besides the HPC command lines parameter, the HPC service has input parameters such as number of nodes and cores to be used by the job, license required by the user to be allowed to submit the job, and maximum execution time used to limit costs and handle non-converging simulations.

In order to provide good user experience, the HPC service allows the application to provide HTML based progress information to the user. The progress reports provided by currently integrated services range from simple progress bars to complete web pages, including embedded 3D rendered models and plots illustrating the convergence of the computation. The latter example execute unmodified software, and generate the HTML based on the log files from an ongoing simulation. The HPC service continuously monitors a pre-defined status file within the application's working directory, and the content of this file is displayed to the user through the browser. It is up to the application provider to fill this file with meaningful content.

When the application is finished, it uploads any output files to the cloud storage. The name of the output files, however, and other output parameters needed by services later in the workflow, are reported back to the HPC Service, and given as service output parameters. Since a HPC job can have any number of output parameters, outputs are supported through a similar design as input parameters. When a job is finished, the HPC service reads a pre-defined result file in the job's working directory. The content of this file is passed as output from the HPC service as a single string. It

is then up to the application provider to first write all output parameters to this file at the end of the job execution, and then provide an application specific post-processing service. The post-processing service should take the output from the HPC service as input, and parse it to separate the different output parameters. The application specific output parameters can then be given semantic information and be passed to any other subsequent service.

The application specific pre- and post-processing services are naturally related to each other, and they only make sense together with the generic HPC service. By modelling these three services as a sub-workflow, the HPC job can be added to any other workflow as a single block with input and output parameters natural to the application. The HPC specific parameters (number of nodes, number of cores, maximum execution time) can either be hardcoded within the HPC application sub-workflow, dynamically defined in the pre-processing service, or user defined through an application consisting of a web form. If they are defined in any other way than by the user, the user experience for a HPC job will be similar to a simulation running in the cloud environment through a regular asynchronous service.

Slow data transfers can potentially introduce performance bottlenecks for HPC jobs. Since each application downloads input data from the cloud storage, and since GSS (Section III-D) is used to handle the file transfers, the software vendors and HPC centers cannot provide restrictions on which cloud storage the users are using. In order to maximize performance, it is recommended that the users make use of the storage solution that is closest to the HPC centers hosting their HPC job. Currently, both of the existing HPC centers in CloudFlow have available cloud storage solutions that are connected to their HPC cluster with high-speed network. Since one of the centers is the host of a private CloudFlow installation, their cloud storage is the only storage available for their users. For the HPC center in the public CloudFlow installation, the closest cloud storage is also the default storage solution for all users. In the current installations, the cost of file transfers to the HPC cluster is similar to the transfer cost between virtual machines in the cloud and the cloud storages. Therefore, data transfers between the cloud and HPC environment do not impose a bottleneck for the users of CloudFlow.

Even though the HPC service is generic, the application that is executed through it will be the part of a software package (as mentioned in Section III-B). The name of the software package that it is part of will therefore be hardcoded as input to the HPC service within the application sub-workflow. This information is then used by the service to check with the Resource Monitoring component that the user has a license to run the application, and to ensure that the software vendor receives the correct license fees. Other resource management tasks, such as reporting CPU hours spent on the computation, are also reported from within the generic HPC service.

To conclude this section, the HPC service facilitates that computationally demanding applications can be executed within a HPC environment, with an interface consisting of semantically described input and output parameters, allowing it to be part of a larger workflow. The service also supports application specific progress reports to be presented to the user during execution, opening for a well informed user experience.

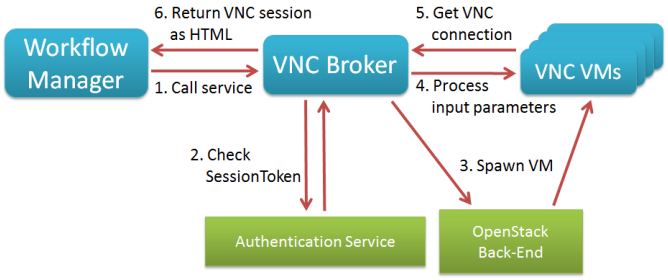

Figure 7. The design of the CloudFlow VNC Service. The VNC Broker handles the requests from the Workflow Manager and manages a set of virtual machines on which the actual desktop applications and the VNC servers run. Input and output parameters are forwarded between the Workflow Manager and the individual VMs.

F. Remote Desktop Applications

Existing engineering desktop applications have typically been developed over several years, and offer their functionality through complex graphical user interfaces. Providing such an application as a CloudFlow web application by re-implementing its complete user interface in a web-based manner, would take a lot of effort and would not be reasonable. Therefore, the CloudFlow Infrastructure offers the possibility to integrate such software directly through a remote desktop concept realized by using the VNC technology [9]. This integration is made through the *VNC Service*. The VNC Service can be used in a workflow as a building block to grant remote access to complex desktop applications. Instead of re-implementing the application's user interface with HTML/JavaScript, the application can be hosted as-is on a virtual machine (VM) inside the cloud environment and can be accessed through a VNC connection. The software providers only have to prepare dedicated VM images for their individual applications.

The CloudFlow VNC Service consists of two parts:

1) A VNC management service, the so-called *VNC Broker*, as part of the core infrastructure.
2) The individual virtual machines that host the actual software and are spawned and despawned by the VNC Broker on demand.

The VNC Broker is a CloudFlow service that interacts with Workflow Manager and manages all incoming VNC requests. Upon request, it spawns a new VM from a prepared image via CloudFlow's OpenStack back-end, and establishes a VNC connection to it. This VNC connection is then converted into an HTML canvas and forwarded to Workflow Manager to be shown in the CloudFlow Portal. When the connection is set up and the user receives the VNC content through the Portal, all communication is done directly between the user's client machine and the VM hosting the desktop application. No intermediate stages are involved. The back-end is realized with an installation of Guacamole [19]. A Guacamole VNC server is installed and configured on the VNC Broker and connects to every individual VNC VM. The mapping of the IP address and port number is done dynamically for each new VM based on the parameters returned from OpenStack. Authentication is handled using the CloudFlow session token passed along with the request from Workflow Manager. The overall architecture of the VNC Service is shown in Figure 7.

The VNC Broker also handles input and output parameters to and from the VNC session. In contrast to standard CloudFlow services, the VNC Service suffers from a similar problem as the HPC integration, as input and output parameters have not only to be passed from the workflow to the service, but all the way through the VNC Broker into the individual virtual machines. To solve this problem, the VMs are equipped with an additional custom SOAP service that is called by the VNC Broker to pass these parameters. If there are more than one input parameter, all inputs have to be serialized into one and deserialized again inside the VM, similarly as for the HPC preprocessing service. Most common input and output parameters however, will be files or folders defined by their GSS file IDs. Therefore, the service on the VM includes the functionality to download files or folders via GSS, and upload results to a dedicated folder in the cloud storage via GSS as well. Once this pre-processing is completed, the service launches the desktop application with optionally provided command line parameters. As the CloudFlow VNC Service is a generic service, billing has to be done individually, depending on which VM image that has been spawned. The total uptime of the VM, as well as usage and license costs of the desktop application, will be taken into account and assigned correctly to the associated software provider.

As mentioned in Section II, network bandwidth and latency are the key factors when operating a VNC session. To reduce latency as much as possible within the infrastructure, we use a direct communication between the virtual machine and the end user. However, in the current setup, spawning the virtual machine upon request is the biggest performance bottleneck. Detailed performance measurements, as well as an in-depth analysis of the additional overhead induced by using an HTML solution instead of a dedicated VNC client application, will be looked into at a later stage.

IV. Validation

The development of the CloudFlow Infrastructure is organized to meet the requirements from end users in manufacturing industries and their software providers. This has given us the opportunity to arrange validations where the end users test the platform and the deployed software against these requirements.

A. Validation methods

To facilitate development and validation, three *waves of experiments* have been set up. In the first wave of experiments, all workflows were tailored towards the needs of one end user in hydropower engineering. Software from six different independent software vendors were integrated with the infrastructure and accessible through the cloud solution, and validated with one common end user. For the second and third wave, European software vendors and end users were invited to test the infrastructure and develop new workflows based on the needs of the end user. In total 14 new experiments were selected, each with one new end user.

At the beginning of each wave of experiments, user requirements analysis was conducted. The user requirements were gathered from the end user, software vendor, and HPC provider, each being a stakeholder of the experiment. Initially, the experiment description provided by each experiment was analyzed, filtered and transformed into a first set of user requirements. Then, during group discussion sessions, experiment stakeholders were invited to confirm, add or remove, and prioritize their user requirements. They were also encouraged to provide success criteria and methods for measuring the success that contributed to experiments validation criteria. In parallel, a discussion related to which services to design and to combine to workflows in order to address the requirements for all of the experiments was also performed. In addition to this, the software vendor in each experiment developed business plans for how to realize the economical potential benefiting both the end user and software vendors. This way, not only the theoretical potential of the software platform is verified, but also that the final solution can sustain as an attractive option.

Formative and summative evaluations were conducted for each experiment. The formative evaluation, which was done remotely, was aimed to fine-tune the development of experiment applications to ensure that the final experiment applications met user requirements. Two activities were performed as part of formative evaluation, i.e., 1) heuristic evaluation by Human-Computer Interactions (HCI) experts to analyze the usability of experiment applications, and 2) assessment of how user requirements were met by the current state of experiment applications. Summative evaluation, which was conducted at end users sites, was aimed to provide final assessment of experiment applications. Four activities were performed as part of summative evaluation, i.e., 1) usability evaluation by end users, 2) recommendation by HCI experts to improve the usability of experiment applications, 3) assessment of how the user requirements were met by the final version of experiment applications, and 4) interview with end users on various aspects of CloudFlow Infrastructure and experiment applications. As part of usability evaluation by end users, they were required to provide complex engineering problems to solve in order to test the extent of experiment application and CloudFlow Infrastructure technical capabilities. The two stages of evaluation meant that the success of both CloudFlow Infrastructure and experiments were validated.

B. Validation results

The results of user requirements revealed that the motivation to use the CloudFlow Infrastructure from end users' point of view varies among the experiments, including attracting new customers, reducing license cost, reducing time spent to create a new product and improving the design of new products. On the other hand, it was also found that there was an overall common goal between software vendors and HPC providers, i.e., to enhance availability of easy to use software and computational resources through

- user friendly interfaces, and
- easy access to advanced computing resources.

For end-users, the user requirements were then extracted to identify metrics that can be used to measure and evaluate the performance of the CloudFlow Infrastructure, e.g., speed, accuracy and usability.

The assessment of how user requirements were met from the 13 already finished and seven ongoing experiments have been processed and shows that the main goals of all experiments have been reached. More importantly, the results showed that end user requirements in each wave were successfully achieved. For instance, in the second wave of experiments, which was composed of seven experiments and involved seven different end users, 30 of 32 (94%) of end user requirements were met, as shown in the left image of Figure 8. The remaining of the user requirements were partially met, and

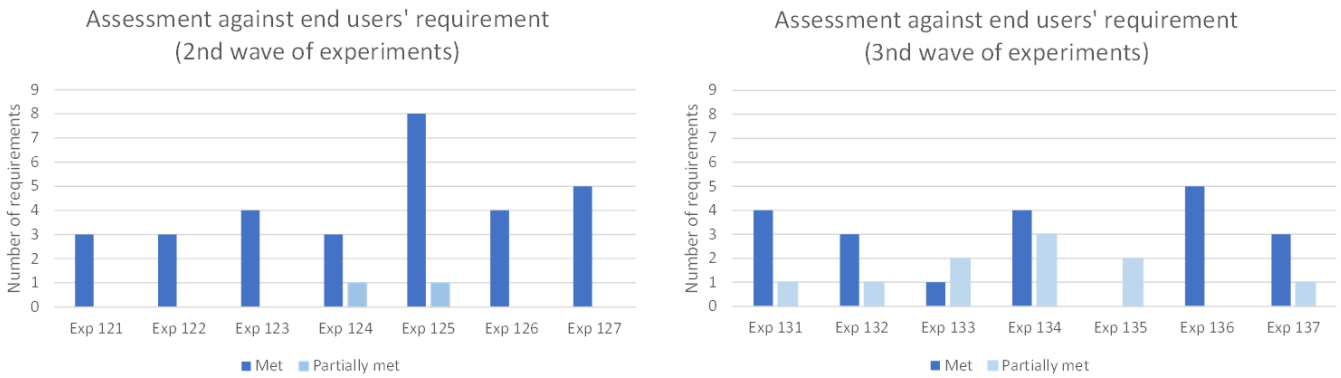

Figure 8. Assessments of end users' requirements for the second wave experiments (left) and third wave experiments (right).

did not represent any critical weaknesses in the infrastructure. Meanwhile, in the third wave of experiments, which was also composed of seven experiments and involved eight different end users, 66.7% of end user requirements were met, as shown in the right image of Figure 8. While this figure was much lower compared to the second wave experiments, on a closer observation, this was due to the fact that the assessment of how user requirements were met is still an ongoing process.

Furthermore, the results of interviews with the end users showed that they would benefit economically from using more computing resources than today, and that the CloudFlow Infrastructure can make it economically feasible. For instance, in the first wave of experiments, by using the experiment applications provided on the CloudFlow Infrastructure, the end user (a hydropower manufacturing SME) has achieved, among others, the following:

- A substantial reduction of time spent (89% faster) on quality check of sub-assembly parts.
- An improvement of the accuracy of the calculated results.

Other examples of economic benefits by end users from the second and third waves can be found in more details on CloudFlow project website [20].

To summarize, the validation results revealed that the concept is functional and makes it more attractive to use cloud computing for various stakeholders. The biggest benefits were found where computationally intensive workflows are presented as engineering apps, that models the expert knowledge for the very concrete artefacts within a unified web-based user interface. These apps allow end users to perform operations that would otherwise require computational resources and human specialists that they do not have access to in-house.

V. Example - Quality Assurance in the Cloud

This section presents a workflow designed to span the functionalities of the CloudFlow Infrastructure. The workflow was originally a first wave experiment, and has been expanded throughout the development of the platform. The requirements are provided by the original end user, who also evaluated the workflow. The workflow is used to align and compare a 3D scan of a model to its original CAD model. Here, the model is a turbine blade used in a hydro power plant. The goal is to confirm that the turbine blade is manufactured according to its design within given tolerances, and later to control wear on the blade after it has been in production for some time.

Before the development of the quality assurance workflow took place, the end user put forward the following five requirements to the workflow:

1) The remote rendering of CAD models and point clouds needs to be provided with low latency, and with at least a frame rate of 15 frames per second.
2) The workflow and services within it must be easy to use.
3) Both experts and non-experts needs to be able to run the workflow. Complex functionality should therefore be hidden, but still accessible for expert users.
4) Improve the engineering process by providing new information not accessible for the end user prior to CloudFlow.
5) Time reduction for the quality assurance process.

Prior to the CloudFlow quality assurance workflow, the end user aligned the CAD model and point cloud manually. This process was prone to error and the result was subject to quality variations.

The quality assurance process consists mainly of three steps, where each step contains one or several CloudFlow services. The main challenge is to align properly the 3D scan data to the CAD model, which usually is a tedious manual process. A fully automated alignment is not necessarily feasible, especially if the CAD model has symmetries. The first step is therefore a coarse manual alignment, which is performed before an automated alignment process, where an optimization process iterates to make the point cloud fit as close as possible to the surfaces of the CAD model. Thereafter, the result of the alignment is reported to the user in an informative and user-friendly manner. The entire workflow, and the four software packages within it, is shown in Figure 9.

A. File selection and conversion

The workflow starts by letting the user choose the CAD model, 3D point cloud, and location for where to store the results from the alignment and distance computations. This functionality is covered by the File Browser, mentioned in Section III-D.

Since CAD models can be stored in different file formats, and since the services later in the workflow expect a predefined file format, a file conversion might be needed at this point. The branching functionality of Workflow Manager described in Section III-A is used at this point, and a conversion service is triggered if the chosen file is not on the

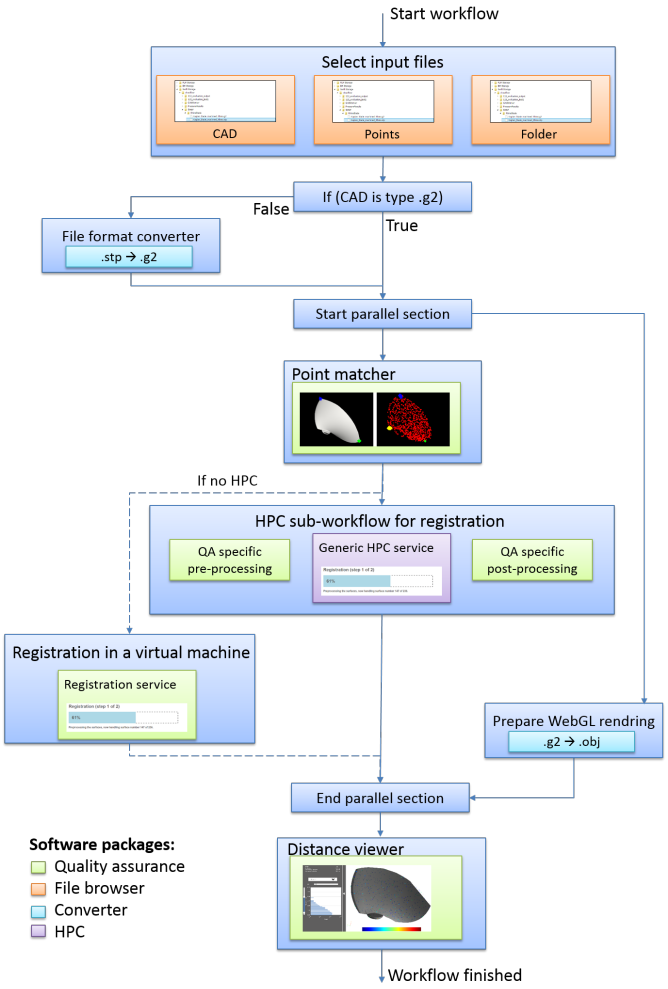

Figure 9. A workflow for quality assurance where a 3D scan of a produced product is compared with the initial CAD model.

pre-defined file format. If triggered, the conversion service accesses a dedicate virtual machine in the cloud which runs the conversion. The conversion service is not guaranteed to finish within a HTTP time out, and is therefore implemented as an asynchronous service. The service launches the conversion as a background process, and Workflow Manager polls on the service to check if the process is finished or not. Before the conversion finishes, it provides a progress bar along with a text describing the current status. Note that this conversion requires no interaction from the user, allowing Workflow Manager to proceed automatically to the next step of the workflow after the conversion is completed.

B. Coarse alignment

The manual alignment step, where the starting point for the automatic alignment is made, is a web application. Here, the CAD model and the point cloud are shown in separate canvases and the user selects corresponding points from the two models. Since the models can be quite large, and to meet the first requirement set by the end user, a hybrid rendering is done through the Tinia framework [11]. The models are rendered server side, generating 3D images that are sent to the web client. The user can freely interact with the local model for an interactive experience, without transferring the CAD model. A drop-down menu is available for expert users to choose rendering configuration manually, according to requirement number 3. Using these configurations can improve the rendering quality, but is not required for a good user experience. Similarly to the file browser, Workflow Manager awaits a message from the client to proceed in the workflow, and this signal is sent when the user has completed the coarse alignment.

C. Automatic alignment

The automatic alignment is computationally intensive and to get the best performance, it executes in the HPC environment through the HPC service described in Section III-E. An application specific pre-processing service is implemented to generate the set of command lines required to run the alignment based on the file IDs obtained in the file chooser applications, converter and coarse alignment application. A post-processing service for the alignment is also implemented in order to enable semantic descriptions to the result from the HPC service. In this case, it will be the file ID holding the aligned point cloud together with the pointwise distance to the CAD model. These three services are then stored as an automatic alignment sub-workflow, hiding any complexities from the HPC service.

The automatic alignment has also been implemented as a single asynchronous service, where the alignment is run in the cloud environment instead of on the HPC cluster. Since the alignment HPC sub-workflow has application specific inputs and outputs, the sub-workflow can be exchanged with the single cloud service directly. This can be useful as a cheaper alternative for users who do not prioritize performance. The service or the HPC block could potentially also be chosen dynamically, using the branching functionality in Workflow Manager. It would even be possible to send the computation to another cloud / HPC provider if the chosen one has limited capacity.

D. Distance visualization

The results after the alignment process are visualized by a WebGL application showing both the CAD model and the aligned point cloud in the same view. Since the browser has limited capabilities, it does not receive the CAD model itself, but rather an approximation more suitable for web rendering. Since the approximation task is independent of all other steps of the workflow, it can be executed in a parallel background service, significantly reducing the start-up time for the post-processing service.

The distance between the models is illustrated both trough statistical information and color-coding of the point cloud. Through the quality assurance approach prior to CloudFlow, the end user had no access to statistical information on the difference between the CAD and the point cloud. This statistical information is therefore provided to meet end user requirement number 4. The user will typically view and inspect these results and take screenshots for documentation before exiting the application. As there are no more next steps, the workflow is completed with a list of workflow output parameters. This list can be accessed later through the user's list of finished workflows.

E. Validation of the quality assurance workflow

The validation showed that all five requirements listed in the beginning of this section was met.

1) Rendering of CAD models was achieved with low latency and acceptable frame rate through hybrid rendering with Tinia, and WebGL.
2) The user interaction was limited to selecting files and destination folder for workflow results, making the coarse manual alignment, and inspecting the result. All these steps provide good user experience, and end users can easily use the quality assurance workflow without any guidance. The workflow is therefore considered user friendly and easy to use.
3) Optional menus for improving rendering quality is available for expert users.
4) Enhanced statistical information was provided to the end user.
5) Following the use of quality assurance application, the end user reported that the quality assurance application reduced the processing time for the alignment from 3 hours to less than 20 minutes.

The quality of the alignment was also improved by at least 10% when compared to the end user's existing approach, which involved many manual steps.

VI. Conclusion and Future Work

In this paper, we have presented the CloudFlow Infrastructure where software packages from different vendors can combined into seamless workflows that are offered to engineering end users. It supports, and has the ability to combine, different HPC and cloud providers, and takes a generic/unified approach to cloud storage solutions and authentication to ensure interoperability. The generic HPC Service allows computationally intensive software to take advantage of HPC resources, while still being an integrated part of workflows and providing good user experience. Similarly, a generic service is presented for providing access to remote desktop applications, through the VNC Service. Complex workflow design is supported through the workflow editor, where conditional branches, loops and parallel service execution can be modelled. The functionality to track and monitor resource consumption by end users, enables that the workflows offered in the infrastructure can be commercialized and open up new business models for the companies involved.

The next steps for the CloudFlow Infrastructure will be to validate the newest additions of the infrastructure. This includes validation of the VNC Service, and the more advanced flow control for workflows. The successful validation shows that the infrastructure is viable, and it is therefore natural to extend CloudFlow to support more cloud providers and HPC centers, and increase the amount and complexity of provided workflows. Another interesting future extension is to combine data streams directly from the factory floors into CloudFlow services, according to Industry 4.0.

We have demonstrated that the CloudFlow Infrastructure can open up the world of advanced multi-vendor software solutions for engineering SMEs, who can not afford to host a computing infrastructure in-house. The use of generic solutions for handling data and utilizing HPC resources, shows the flexibility of our approach, and makes it easy for software vendors to integrate their software in a cloud environment. CloudFlow provides a new distribution channel for the software vendors where they can offer their software based on new cloud-based business models, either as a pay-per-use license, or by periodic licenses.

Acknowledgement

This research was conducted in the context of the CloudFlow project, which is co-funded by the 7th Framework Program of the European Union, project number 609100. More information and news about CloudFlow can be found on the project website at `http://eu-cloudflow.eu/`.

References

[1] 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.

[2] 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, March 2015, pp. 1520–1525.

[3] SimScale. Website, retrieved: 2017-05-29. [Online]. Available: https://www.simscale.com/ (2017)

[4] cloudSME, simulation for manufacturing & engineering. Seventh Framework Programme (FP7) under grant agreement number 608886. Website, retrieved: 2017-05-29. [Online]. Available: http://www.cloudsme-apps.com/ (2017)

[5] S. J. E. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "Cloud computing for simulation in manufacturing and engineering: Introducing the cloudsme simulation platform," in Proceedings of the 2014 Annual Simulation Symposium, ser. ANSS '14. San Diego, CA, USA: Society for Computer Simulation International, 2014, pp. 12:1–12:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2664292.2664304

[6] Fortissimo factories of the future resources, technology, infrastructure and services for simulation and modelling. Seventh Framework Programme (FP7) under grant agreement number 609029. Website, retrieved: 2017-05-29. [Online]. Available: https://www.fortissimo-project.eu/ (2017)

[7] B. Koller, N. Struckmann, J. Buchholz, and M. Gienger, "Towards an environment to deliver high performance computing to small and medium enterprises," in Sustained Simulation Performance 2015. Springer International Publishing, 2015, pp. 41–50.

[8] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications," in UCC 2014: 7th IEEE/ACM International Conference on Utility and Cloud Computing, R. Bilof, Ed. IEEE Computer Society, 2014, pp. 269–277.

[9] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," IEEE Internet Computing, vol. 2, no. 1, 1998, pp. 33–38.

[10] Microsoft. Remote desktop protocol. Website, retrieved 2017-02-15. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx (2017)

[11] C. Dyken, K. O. Lye, J. Seland, E. W. Bjønnes, J. M. Hjelmervik, J. O. Nygaard, and T. R. Hagen, "A framework for opengl client-server rendering," in 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan. IEEE Computer Society, 2012, pp. 729–734. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2012.6427506

[12] C. Altenhofen, A. Dietrich, A. Stork, and D. Fellner, "Rixels: Towards secure interactive 3d graphics in engineering clouds," Transactions on Internet Research (TIR), vol. 12, no. 1, Jan. 2016, pp. 31–38.

[13] 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

[14] M. A. Aslam, S. Auer, J. Shen, and M. Hermann, "Expressing business process model as owl-s ontologies," in Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer based Workflows (GPWW 2006), 2006, 4th International Conference on Business Process Management (BPM 2006), Vienna, Austria, September 2006.

[15] Oracle, "Oracle BPEL process manager datasheet," 2009, [retrieved: 2017-05-29]. [Online]. Available: http://www.oracle.com/technetwork/middleware/bpel/overview/ds-bpel-11gr1-1-134826.pdf

[16] R. Cyganiak, D. Wood, and M. Lanthaler, "Web Services Architecture," W3C Working Group Note, 2004, [retrieved: 2017-05-29]. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[17] V. Gezer and S. Bergweiler, "Service and workflow engineering based on semantic web technologies," in UBICOMM 2016: The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies. IARIA, October 2016, pp. 152–157.

[18] Openstack Keystone. Website, retrieved: 2017-05-29. [Online]. Available: http://docs.openstack.org/developer/keystone/ (2017)

[19] The Apache Software Foundation (ASF). Guacamole VNC. Website, retrieved 2017-05-29. [Online]. Available: http://guacamole.incubator.apache.org/ (2017)

[20] CloudFlow: Computational cloud services and workflows for agile engineering. Seventh Framework Programme (FP7) under grant agreement number 609100. Website, retrieved: 2017-05-29. [Online]. Available: http://eu-cloudflow.eu/ (2017)