Adaptive and Reflective Middleware for the Cloudification of Simulation & Optimization Workflows

Emad Heydari Beni, Bert Lagaisse, Wouter Joosen
imec-DistriNet, KU Leuven, 3001 Leuven, Belgium
{emad.heydaribeni, bert.lagaisse, wouter.joosen}@cs.kuleuven.be

ABSTRACT

The simulation and optimization of complex engineering designs in automotive or aerospace involves multiple mathematical tools, long-running workflows and resource-intensive computations on distributed infrastructures.

Finding the optimal deployment in terms of task distribution, parallelization, collocation and resource assignment for each execution is a step-wise process involving both human input with domain-specific knowledge about the tools as well as the acquisition of new knowledge based on the actual execution history.

In this paper, we present motivating scenarios as well as an architecture for adaptive and reflective middleware that supports smart cloud-based deployment and execution of engineering workflows.

This middleware supports deep inspection of the workflow task structure and execution, as well as of the very specific mathematical tools, their executions and used parameters. The reflective capabilities are based on multiple meta-models to reflect workflow structure, deployment, execution and resources. Adaptive deployment is driven by both human input as meta-data annotations as well as the actual execution history of the workflows.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; H.1 [**Models and Principles**]: Miscellaneous

Keywords

Workflow, Cloud, Middleware, Adaptability, Reflection

1. INTRODUCTION

Engineers in major industries, such as aerospace and automotive, use simulation and optimization workflows to create, simulate and optimize complex designs. Such workflows are complex and long running processes, which are typically composed of various software tools and services, to simulate and optimize physical properties such as strength, vibrations, geometrical decomposition or material selection. Engineers use different hardware to execute these workflows, e.g. their desktop computers or High Performance Computing (HPC) clusters.

Current situation. Desktop computers have limited capacity in terms of processors, memory and storage. In addition, the parallel execution of the experiments is tied to the number of available computers. HPC clusters, unlike desktop computers, are very efficient and powerful, but they are constructed with dedicated expensive hardware and their capacity is still limited. Besides, time slot reservation and complex queuing API are yet another hassle for those with long-running or recurring experiments.

A quick example. For example, civil engineers employ evolutionary optimisation algorithms to minimise the mass and maximum stress of truss bridges. To execute these workflows, they need to have lots of resources in place such as sufficient processing power, memory, storage and network connectivity, as well as necessary set of analysis tools. To speed up the process, parts (or all) of the workflows should be arranged to be performed in parallel, meaning that multiple instances of tools as well as multiple servers and networking resources are needed to be available beforehand. Last but not least, truss analysis is executed recurrently with different parameters. Engineers typically change the algorithms and parameters, and execute the experiments multiple times to obtain a desirable output. These variations might require the adaptation of the underpinning infrastructure and workflow deployment.

The promise of the cloud. Engineers can nowadays benefit from cloud computing to gain on-demand access to the required resources for their workflows, often based on cheap commodity hardware. Cloud computing is a model for enabling *on-demand* network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) [9]. Configuration management and cloud orchestration are also useful emerging paradigms in this domain. Software configuration management tools enable practitioners to specify configurations, configuration items and other attributes of a server and all of the required software. Cloud orchestrations empower clients to compose architecture, tools, processes, and their connections as workflows in order to provision the required cloud-related resources such as virtual machines, virtual networks, and required infrastructure software and middleware platforms. As such, the infrastructure and deployment process become completely *composable and programmable.*

ARM'17, December 11–15, 2017, Las Vegas, NV, USA
 ISBN 978-1-4503-5168-3/17/12 ...$15.00
DOI: https://doi.org/10.1145/3152881.3152883.

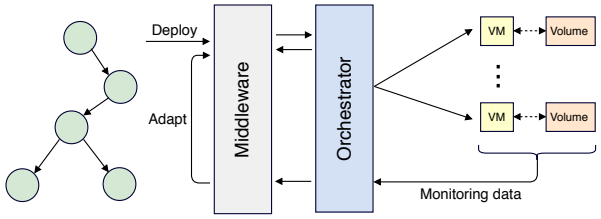

Figure 1: Middleware for cloudifying simulation workflows

Challenges. There are still key problems and challenges when deploying and executing engineering workflows in the cloud. One needs to automate the deployment, as well as support smart scaling and execution of simulation and optimization workflows in the cloud. For each deployment and execution of the workflow, this process includes adaptive deployment to collocate, separate and parallelize the different tasks and their specific tools on the right amount and the right type of nodes. Both can also variate depending on the specific parameters for a certain execution. Automating this process includes:

- Automatic determination of required resource types in the private or public cloud, e.g., virtual machines, operating systems, storage volumes, virtual network, etc.
- Automatic estimation of the amount of cloud resources, e.g., number of virtual machines and processors, amount of memory, storage capacity, etc.
- Automatic bootstrapping of required infrastructure and deployment of analysis software and tools.

InfraComposer: towards smart deployment in the cloud. To address these challenges, we present a **reflective and adaptive middleware** that enables and manages smart, adaptive workflow deployment, scaling and execution in the cloud. We leverage both the domain-specific knowledge about the concrete tools that are used, and deep inspection of these tools when deployed and executing on the cloud platform.

The **adaptive middleware** is driven by both input from the engineers about the properties of the tools they use, as well as execution history of these tools. The input from the engineers is specified as annotations on the workflows, and is based on human knowledge and assumption about the tools with regards to CPU usage, memory usage and network usage. The execution history over time will be used to optimize the original deployment and scaling plan, and thus to further adapt to actual real execution knowledge (see Fig. 1).

As such, our middleware defines two key contributions to existing orchestration and deployment middleware:

1. *Annotation-driven resource reservation and deployment planning.* Based on the annotations in the workflow, an initial deployment plan will be generated. A deployment plan is a topology and orchestration specification for a cloud application which can be executed by a cloud orchestrator.
2. *History-driven predictive scaling and reconfiguration.* In addition, the middleware adapts the configurations of the deployment plans based on the execution history of the workflows using previous results. This should be done based on statistical analysis of the execution history, or even more advanced machine learning tactics such as predictive algorithms or clustering.

To achieve this, the **reflective meta-models in the middleware** enable reification of

1. key architectural concepts such as workflows, tasks[1], specific tools and their deployment,
2. key execution concepts such as specific tool executions with specific parameters, and
3. key resource utilisation concepts such as nodes, cores, cpu time, memory, storage and network metrics.

The rest of this paper is structured as follows. Section 2 presents two motivating scenarios of engineering workflows and their common patterns. Section 3 describes the architecture and the concepts of the middleware. Section 4 validates multiple adaptive (re)deployment scenarios. Section 5 presents the state-of-the-art in cloudification of workflows, adaptive middleware, and auto-scaling techniques. Section 6 concludes this paper and outlines our research outcomes and future work.

2. MOTIVATION AND USE CASES

Engineers usually run workflows many times with different parameters. In this section, we describe two examples of such workflows from different domains.

Truss structural analysis.

Civil engineers typically perform multi-objective optimisation of truss structural analysis to accomplish various goals by using an evolutionary optimisation algorithm, e.g. to minimise mass and maximum stress. This workflow consists of tools such as Truss Dynamic and Truss Static to simulate and optimize the strength and size of the bridge.

Motivating Scenario. When considering the tasks in the workflow, it is unclear how Truss Dynamic and Truss Static should be deployed: on a single node or on different nodes. The most optimal deployment is achieved in a step-wise adaptive process, first driven by the engineer's annotations, then by the execution history of the tool.

(i) First, the engineer of the workflow specifies that both Truss Dynamic and Truss Static are disk-intensive tools because they read and write large files from disk, and they communicate using these files on disk. At least, this is the engineer's assumption. He defines this assumed knowledge as annotations. (ii) The middleware will collocate the two tools and all parallel executions on one virtual node in the cloud and execute them as specified in the workflow. (iii) However, execution history shows that the tools are mostly CPU intensive. In the actual deployment environment, the directory used to store and read files is a high-bandwidth network attached storage device, which reduces IO latency and throughput of the workflow compared to the limited throughput of a virtual drive on a local disk. As such, the execution of whole workflow on one node becomes CPU bound rather than IO bound. (iv) The deployment plan should be updated to distribute the different executions over multiple virtual machines.

Electrical wiring interconnection system design.

Another multidisciplinary design optimisation (MDO) example can be found in the design process of aircrafts. One of the important steps of this process focuses on electrical

[1]We use the terms *tasks* and *activities* interchangeably.

wiring interconnection system (EWIS) design. Engineers design and execute complex wire routing simulation experiments to find an optimised solution.

Motivating scenario. This workflow employs a very large amount of data of physical features of an aircraft as input which results in lots of parallel runs. Therefore, the number of nodes and tools instances are unclear to an engineer.

(i) First, the engineer of the workflow specifies five instances of the nodes, and an expected execution time. (ii) The middleware will instantiate the nodes and execute the experiment as specified in the workflow. (iii) However, the execution history shows that the execution took much longer than the initial anticipation because of the large number of parallel runs and scheduled jobs. (iv) The deployment plan should be updated to adjust the number of nodes (auto-scaling) with regards to the expected execution time.

Section 4 presents more scenarios. Based on each of these workflows, a similar pattern emerges:

- The deployment middleware needs initial domain knowledge about the tools to achieve a first deployment plan.
- Actual executions of the tool with specific parameters might require optimisation of the deployment plan.
- Most of these workflows need different discipline analysis tools for execution, and each tool could be installed on a specific operating system and specific host type (memory-focused host, CPU-focused host or high-performing storage hosts with SSDs).
- A lot of these workflows are computationally intensive, and their execution sometimes takes hours, days or weeks to be completed. Engineers use parallel runs to speed up the execution. That may have impact on network topology too.
- Most of these workflows are involved in a continuous improvement process by reconfiguration of the parameters, and recurring re-executions to fulfil the optimised objectives. As such, the actual parameters of the executions might require adaptation of the deployment as tools might become more dependent on CPU than disk for different parameters.

These common patterns introduce several key problems and challenges (refer to Section 1) leading to manual, duplicate, complex, time-consuming work for engineers.

3. THE INFRACOMPOSER MIDDLEWARE

This section presents InfraComposer, a reflective and adaptive middleware that deploys all cloud and software resources based on the given annotations within engineering workflows.

During execution, it collects runtime information about task executions and the underpinning infrastructure by reflective monitoring of resources, as well as deep inspection of software tools. It therefore enables the middleware to reconfigure the deployment plans predictively to be adaptive for recurrent execution of the workflows based on the execution history.

The InfraComposer middleware consists of four main components to address the aforementioned challenges in Section 1 (see Fig. 2): (i) a *workflow manager* component to expose a workflow deployment API and to identify the annotated tasks and their annotations, (ii) a *configurator* component to generate configurations based on given annotations with respect to execution history data, (iii) a *deployment plan composer* component to produce deployment plans based on elementary deployment modules for the cloud orchestrator and to initiate the deployment, and (iv) a *monitoring* component to store live monitoring data of workflow execution.

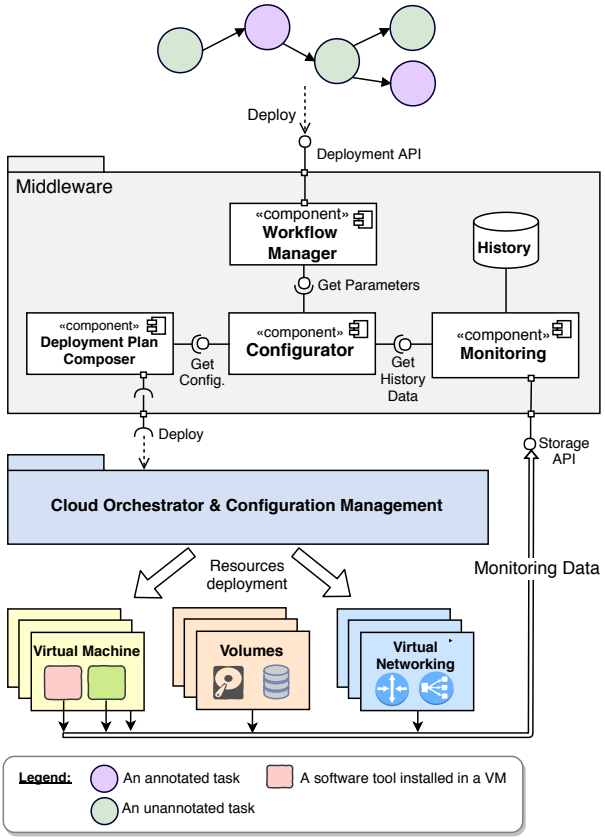

Figure 2: Overview of the proposed middleware with the annotation processing and configuration components.

The rest of this section is structured as follows. First, we describe the annotation-based adaptive deployment. We then elaborate on the reflective capabilities and the meta-models. Third, we conclude with a discussion on adding intelligence and new knowledge to "smart" adaptive middleware.

3.1 Annotation-based Deployment

A simulation and optimisation workflow is a group of tasks that, once completed, will accomplish some objectives. As explained in Section 1, these tasks employ different analysis tools, which are responsible for the execution. Workflows and tasks can be annotated to provide more information about the required resources. InfraComposer is capable of identifying these annotations in the *workflow manager* component to provide necessary information for the *configurator* component. There are two categories of annotations: direct-deployment and resource-consumption annotations.

Direct-deployment annotations. These annotations provide the middleware with direct information concerning deployment of workflows on the cloud. The crucial aspects described by annotations are the employed analysis tools,

Category	Annotation	
Disk	Disk intensive	percentage
	Required disk space	GB
	Data locality	location
Memory	Memory intensive	percentage
	Required RAM	GiB
Processor	CPU intensive	percentage
	GPU intensive	percentage
	Required cores	number of cores
Network	Network intensive	percentage
	Required bandwidth	Mbps

Table 1: Four main categories of resource-consumption annotations.

their deployment locations and number of instances. Tools can either be collocated or separately deployed on the nodes. For example, annotations could describe that some tools should be deployed on one node, others on individual nodes, and there should be five instances of each node. Furthermore, network-related annotations can propose a networking scheme for tools and nodes where necessary. For example, network-level segregation of nodes can be achieved for a network intensive workflow.

Another direct-deployment annotation is the identifier of the existing resources (e.g. instance images, volumes, networking components, etc.). For instance, some of the engineering tools need human involvement during the installation process, or install slowly due to the size of the packages. Therefore, these tools can be installed and prepared as virtual machine images to speed up the deployment process. The unique identifiers of the images help the *configurator* component provide configurations to the *deployment plan composer* component.

Resource-consumption annotations. These annotations provide general, approximate information about the infrastructural resource requirements of the tools with regard to disk, memory, processor and network. The four main categories of resource-consumption annotations are presented in Table 1.

Workflows can specify whether the experiment is disk intensive, as well as the required space. Cloud providers, either private or public, offer various types of storage systems with varying capabilities, speed, etc. In addition, some clients are concerned about data locality due to the enterprise policies or governmental law (e.g. GDPR[4]). Such annotations guide InfraComposer to select appropriate storage options with respect to the given constraints.

The computationally intensive workflows should employ proper virtual machines in order to get executed efficiently and to minimise the interference with other co-existing cloud users. Some virtual machines share the physical processors with other tenants, and some have CPU-pinning, meaning that the virtual cores are mapped to the physical cores in a shared-nothing approach. Besides, some public cloud providers offer domain-specific types of virtual machines such as accelerated instances with GPU.

Furthermore, parallel execution of workflows may have considerable network overhead due to the continuous transfer of large chunks of data. That can easily saturate the bandwidth, slow down the execution, and interfere with other co-existing users. Such annotations enable the middleware to compose appropriate network architectures based on the available networking infrastructure.

3.2 Reflective Capabilities and Meta-models

The middleware follows the temporal correspondence approach into different meta-models [3]. There are four styles of reflection in InfraComposer represented by four meta-models, namely structural, deployment, execution and resource reflection.

Structural Reflection. Structural reflection [5] results in a meta-model of the different static concepts in the workflows defined by the engineers, which represents the structure of workflows and tools within the middleware and the execution history. Fig. 3 illustrates the meta-model. This meta-model describes engineering workflows and composition of activities and tools, as well as annotations.

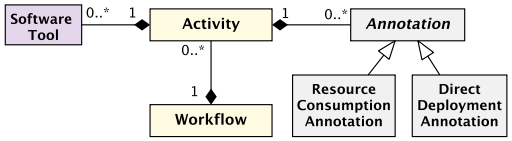

Figure 3: The structural meta-model.

Deployment Reflection. Deployment reflection results in a meta-model representing and reifying the concepts in the deployment model. Fig. 4 illustrates the deployment plan and the mappings between workflows, activities, tools (not illustrated), and cloud-based components such as compute nodes, storage and networking elements. The components are (re)configurable through configuration files by InfraComposer at run-time.

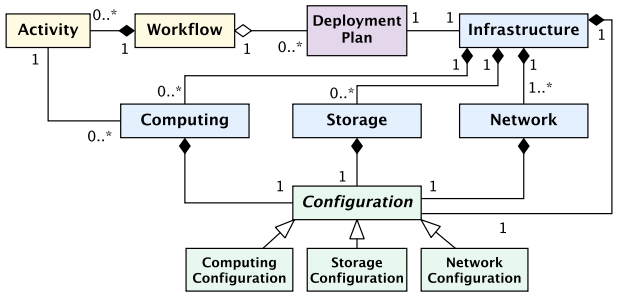

Figure 4: The deployment meta-model.

Execution Reflection. Execution reflection results in a meta-model of the execution of activities and specific tools on specific nodes. This model reifies concepts such as execution per workflow, execution per activity, and execution per tool with respect to the given parameters (see Fig. 5).

Resource Reflection. Resource reflection results in a meta-model of the underpinning cloud infrastructure resources and domain resources (see Fig. 6). Cloud infrastructure resources include concepts such as processing (cores), compute nodes, memory, network and storage, which are reified for each execution by consumption pattern. Such reflection allows the monitoring and adaptation phase to benefit from coarse-grained or deep introspection of resources, leading to more efficient resource allocation in the future execution of the workflows.

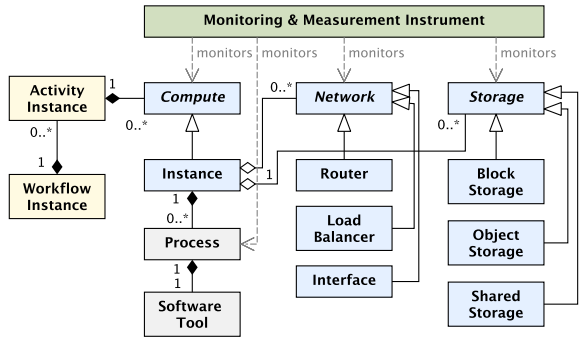

Figure 5: The execution meta-model.

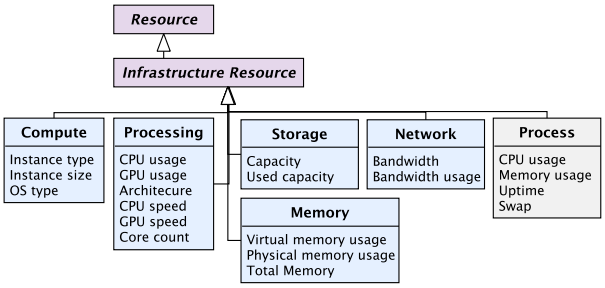

Figure 6: The resource meta-model.

3.3 Towards Smart Adaptation

InfraComposer monitors the execution of the workflows and builds an execution history, with which it fine-tunes the deployment plan and the configurations to make the future executions more efficient. Adding intelligence to the smart adaptation capabilities of the middleware requires the acquisition of new knowledge about the execution of the different tasks and tools. For example, an annotation suggested that a tool was disk intensive, but execution history teaches us that it is CPU intensive and only uses two cores for input files smaller than 100 MB.

InfraComposer is a (self-)adaptive middleware following the MAPE-K [6] architecture of self-adaptive systems [8].

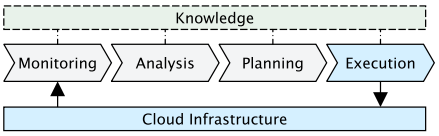

Figure 7: Adaptive middleware using MAPE-K control loop.

The *monitoring* phase collects information from the cloud and workflow-specific resources. The *analysis* phase uses the persisted monitoring data to assess the previous executions and to determine potential future improvements. The *planning* phase reconfigures the deployment plans, using the history-driven predictive scaling propositions produced in the previous phase. The *execution* phase redeploys the cloud resources and the software tools for another execution. The *knowledge* about the workflows, and the cloud infrastructure (i.e. public/private providers, available types of resources, the existing resources, etc.) is a cross-cutting aspect serving information to each phase.

For smart adaptation and auto-scaling of resources, the analysis and the planning phase are crucial. Auto-scaling techniques are categorised into two classes: *reactive* and *proactive* [7]. Reactive techniques are concerned with the adjustment of deployment plans and resource allocation based on certain thresholds, and proactive approaches attempt to predict and anticipate the efficient configurations.

Static policy-based techniques are an example of the practical reactive techniques. For example, if a processing metric driven by statistics based on the execution history exceeds a particular threshold defined in the rules, showing that the activity is a computationally intensive task, the configurator could reconfigure the deployment plan for that node to employ CPU-focused virtual machines.

The key open challenge is finding the right applicable method to detect this additional knowledge from the execution history.

- Basic statistical analysis, such as the average execution time of a certain tool.
- Machine learning, such as clustering techniques to try and detect the distinguishing features in the execution history.
- Deep learning, which can automatically detect distinguishing abstract features in the execution history.

We assume advanced machine learning tactics are needed to detect key differentiating features in the execution history. Finding the right algorithms is part of our future work.

4. IMPLEMENTATION AND VALIDATION

In this section we describe some specific adaptive deployment scenarios and validate how both the reflective capabilities as well as the adaptation capabilities of the middleware can cope with each adaptive deployment scenario. As a validation of the concepts and architecture of InfraComposer, we created a middleware and, on top of it, prototyped the *EWIS* design (refer to Section 2) as our engineering workflow. We employed a policy-based approach for the analysis and the planning phase. We assume that the engineer has made an inappropriate assumption about the workflow and its requirements in each scenario.

Scenario 1. We assume that some activities within the workflow are not annotated to be compute-bound, but the reflective monitoring data shows that the nodes and the tools consume more than a particular threshold. Therefore, the deployment plan is updated to employ CPU-focused virtual machines for those activities (or more cores).

Scenario 2. We assume that some tools are not annotated to be memory intensive, but the reflective monitoring shows that the simulation tools load a very large chunk of data to the memory before and during the processing. Consequently the deployment plan is altered to scale up and to employ memory-focused virtual machines with a larger amount of memory.

Scenario 3. We assume that the workflow is not annotated to be data- or network-intensive, but the reflective monitoring shows that the nodes perform lots of I/O operations on the shared volume. Therefore, the deployment plan is reconfigured to change the network configurations, and to join the nodes to a high-bandwidth network that offers faster shared storage volumes, which is in our case the NetApp storage solution.

Scenario 4. We assume that the workflow is annotated to deploy two of the analysis tools on one node, but the deep reflective monitoring data shows that both tools consume CPU higher than a particular threshold. The deployment plan is then updated towards separate deployment of the tools on individual nodes.

Scenario 5. We assume that some of the tools are anticipated to be deployed cheaply on a few number of instances, but the reflective monitoring shows that there are considerable number of parallel runs (i.e. scheduled job) due to the large size of input parameters and datasets (about the physical features of the aircraft), which results in a long-running overall execution. Consequently the deployment plan is reconfigured to scale out the nodes and to employ more instances of the tools to satisfy the expected execution time requirement.

5. RELATED WORK

Reproducibility and repeatability of workflow execution environment are crucial aspects in scientific and engineering workflow deployment. In that regard, Santana-Perez et al. [12] describe the execution environment of workflows using semantic vocabularies to produce annotated workflows (i.e. logical preservation of execution environment). TOSCA [14] is an OASIS standard to describe the topology of cloud-based applications towards portable, reproducible application deployments. Qasha et al. [10] combine two execution-environment reproducibility techniques (i.e. the logical and physical preservation) of scientific workflows using TOSCA in a container-based approach. In addition to the plain reproducibility concerns, our middleware architecture employs reflection concepts to reconfigure deployment plans, resulting in *efficient* execution environments.

The concept of reflection was first introduced by Smith [13] in programming languages. Other works [1, 11] adopted the concept in the middleware platforms, focusing on reconfigurability and openness of such systems. Blair et al. [1] present two types of reflection: *structural reflection* and *behavioural reflection*. Structural reflection is concerned with the structure and the content of the component, which is represented by two distinct meta-models, namely the encapsulation and composition meta-models. Behavioural reflection is concerned with activity in the system, which is represented by the environmental meta-model. Existing literature [2] extends the architecture by presenting the resource meta-model to address the reification of resource management. Our meta-models are inspired by these studies.

Deployment plans are reconfigured by re-scaling of resources based on execution history, either horizontally or vertically. Lorido et al. [7] categorise auto-scaling techniques into *reactive* (i.e. based on rules and current data) and *proactive* (i.e. based on prediction) approaches, as well as a more fine-grained classification, resulting in: (1) threshold based rules, (2) reinforcement learning, (3) queuing theory, (4) control theory, and (5) time series analysis. Our vision is to employ a hybrid technique, e.g. combining the threshold-based rules with the time series analysis.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced InfraComposer which enables complex simulation and optimisation workflows to have smart deployment and scalable execution on cloud infrastructures. To this end, we proposed a step-wise approach to achieve an optimal deployment plan for such workflows. The approach includes: (i) obtaining engineers' input about initial, direct deployment of workflows through annotations in the first place, and (ii) the acquisition of new knowledge based on the actual execution history employed to produce improved deployments. The adaptive and reflective architecture followed the temporal and ontological correspondence [3] approach into four different meta-models, namely structural, deployment, execution and resource reflection. As a validation of the concepts, the applicability, and the reproducibility of our approach, we presented some specific adaptive deployment scenarios and validated how both the reflective and the adaptation capabilities of the middleware can cope with each scenario.

The future work includes the further validation of InfraComposer in more extensive workflows, as well as conducting comprehensive performance evaluation with regard to deployment adaptations. The other research direction includes usage of additional deployment constraints via annotations, and exploring their impact on the adaptive process of obtaining optimal deployments (e.g. budget constraints).

7. REFERENCES

[1] G. Blair et al. An architecture for next generation middleware. In *Distributed Systems Platforms and Open Distributed Processing*. Springer-Verlag, 1998.

[2] G. Blair et al. The design of a resource-aware reflective middleware architecture. In *Meta-Level Architectures and Reflection*. Springer, 1999.

[3] G. Bracha and D. Ungar. Mirrors: design principles for meta-level facilities of object-oriented programming languages. ACM, 2004.

[4] E. Commission. Regulation eu 2016/67. 2016.

[5] P. Grace et al. A distributed architecture meta-model for self-managed middleware. In *ARM06*. ACM, 2006.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[7] Lorido-Botran et al. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 2014.

[8] F. D. Macías-Escrivá et al. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 2013.

[9] P. Mell et al. The nist definition of cloud. 2011.

[10] R. Qasha et al. A framework for scientific workflow reproducibility in the cloud. In *e-Science*. IEEE, 2016.

[11] M. Roman, F. Kon, R. H. Campbell, et al. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online*, 2(5):1–29, 2001.

[12] I. Santana-Perez et al. Reproducibility of execution environments in computational science using semantics and clouds. *Future Generation Computer Systems*, 2017.

[13] B. C. Smith. *Procedural reflection in programming languages*. PhD thesis, MIT, 1982.

[14] TOSCA-OASIS. Topology and orchestration specification for cloud applications primer v1, 2013.