Infrastructure aware scientific workflows and infrastructure aware workflow managers in science gateways

Peter Kacsuk, Gabor Kecskemeti, Attila Kertesz, Zsolt Nemeth, József Kovács, Zoltán Farkas
Laboratory of Parallel and Distributed Systems
MTA SZTAKI the Institute for Computer Science and Control of the Hungarian Academy of Sciences
Budapest, Hungary
{kacsuk.peter, kecskemeti.gabor, kertesz.attila, nemeth.zsolt, kovacs.jozsef, farkas.zoltan}@sztaki.mta.hu

***Abstract*—The workflow interoperability problem was successfully solved by the SHIWA project if the workflows to be integrated were running in the same grid infrastructure. However, in the more generic case when the workflows were running in different infrastructures the problem has not been solved yet. In the current paper we show a solution for this problem by introducing a new type of workflow called infrastructure-aware workflow. These are scientific workflows extended with new node types that enable the on-the-fly creation and destruction of the required infrastructures in the clouds. The paper shows the semantics of these new types of nodes and workflows and also how they can solve the workflow interoperability problem. The paper also describes how these new type of workflows can be implemented by a new service called Occopus, and how this service can be integrated with the existing SHIWA Simulation Platform services like the WS-PGRADE/gUSE portal to provide the required functionalities of solving the workflow interoperability problem.**

Keywords—workflow; cloud; virtual infrastructure; dynamic deployment.

I. INTRODUCTION

Workflows are getting more and more popular in science user communities in order to formulate complex simulation and modeling activities. Unfortunately, too many different workflow languages and workflow managersoften prevent user communities from reusing workflows created by another user community or sometimes even by the same community. The situation is even worse because most of the workflow systems are tightly connected to one particular distributed computing infrastructure (DCI), i.e., the workflow developed in a certain workflow system can be executed only in the DCI for which the workflow system was designed. It means that if a user community selects a certain workflow system, it is locked into the DCI integrated with the workflow system. This causes problems not only in reusing and sharing workflows among different user communities, but also porting workflow applications from one DCI to another.

This problem was addressed by the EU FP7 SHIWA project [21] that offered two different approaches to solve the problem. The so-called white box model enabled to transform the workflow to be reused into the target workflow system used by the new user community. Although this concept is very flexible and enables any modification of the original workflow, the transformation can be applied only for those workflow structures that are commonly used in all the WF languages. Workflow system specific structures usually prevent the transformation. The other model, the so-called black box model assumes that the workflow to be reused is a black box and must be executed as it is without any internal modification or reconfiguration of the workflow. It makes the reuse very easy but it means that when the workflow is reused by another community it should run in the same concrete DCI where it was developed. However, this is a very hard condition since the new community might not access the original DCI for which the workflow was developed and hence cannot reuse the workflow.

This problem was raised in the EU FP7 ER-Flow project [9] where four different user communities (astrophysics, biomedical, helio-physics and chemistry communities) intensively investigated the usability of the black box concept. As long as they wanted to share and reuse workflows in the same concrete DCI it was working but when they could not access the original concrete DCI they were not able to reuse the workflows.

Fortunately, recently a new DCI type appeared called clouds, where we can dynamically deploy even a complete DCI. This approach can facilitate solving the problem mentioned above. If the TOSCA [25] descriptor of the DCI in which the workflow was developed is available, we can deploy the required DCI in the cloud by means of a TOSCA-compliant cloud orchestrator. After that, we can execute the reusable workflow there. This way we replace the original physical DCI with a virtual infrastructure (VI). The problem is that we need such a TOSCA-compliant cloud orchestrator by which a DCI can be deployed in all the most popular cloud systems like Amazon, OpenStack, OpenNebula, etc. In SZTAKI we have develop the Occopus cloud orchestrator service that enables the required DCI deployment in all the major cloud systems.

There are two options for controlling the deployment activity of the cloud orchestrator service inside a workflow system. The first option is that the user places the required control nodes into the workflow. We will call these kinds of workflows that explicitly contain deployment control nodes as infrastructureaware workflows. The minimum set of such nodes contains two nodes: DEPLOY for deploying the required VI in the cloud and UNDEPLOY to remove the VI from the cloud.

The other option is that the workflow manager can automatically recognize the situation when a new VI should be deployed in the cloud and calls the cloud orchestrator service to deploy the VI. Similarly, the workflow manager can detect that there is no need to further maintain the VI and hence can call the cloud orchestrator service to remove the VI. We will call these kinds of workflow managers as infrastructure aware workflow managers.

In the present paper we investigate both possible concepts showing their possible logical representations and implementations.

The overall structure of the paper is as follows. In Section II we give more details of the SHIWA black box concept and its usage in the ER-Flow project. Then in Section III we investigate the possibilities of creating infrastructure aware workflows. In Section IV, we shortly describe the Occopus service that enables the deployment of complete DCIs in the major cloud systems. This section will also explain the infrastructure aware workflow execution mechanism and how it is integrated into the WS-PGRADE/gUSE gateway framework based on which the SHIWA portal works. Section V introduces the concept of infrastructure aware workflow managers and shows a possible implementation mechanism for them. Section VIgives an overview of related research and finally, in the Conclusions we assess the two concepts and show the status of the current research and its further directions.

II. SHIWA Black Box Concept in Practice

A. SHIWA Simulation Platform

The SHIWA black box concept, realized by the SHIWA Simulation Platform, consists of the following components:

- SHIWA Workflow Repository to store the published reusable workflows (developed and operated by Univ. of Westminster, UK)
- SHIWA Portal to enable the integration and execution of the reusable workflows into the native workflows of the SHIWA Portal (developed by MTA SZTAKI and operated by Univ. of Westminster, UK)
- SHIWA Virtual Organization (VO) that provides a DCI where SHIWA'sreusable workflows were developed and run (services of the VO were run in four countries)

B. SHIWA Workflow Repository

In order to publish the reusable workflows, the SHIWA project developed the SHIWA Workflow Repository [21]. Workflows written in any workflow language can be uploaded here by defining the so-called workflow bundle that contains all the relevant information about the structure of the workflow, the files used or produced by the workflow, version and configuration of the workflow, etc. The workflow bundle also contains details of the workflow implementation and execution environment including the identifier of the concrete DCIs where the workflow nodes can run. The detailed description of the structure of the workflow bundle can be found in [21].

There are two options to define the workflow bundle for a particular workflow. Option 1, the preferred one is where the developer of the workflow system creates an automatic workflow bundle generator and uploader. In this case after completing the workflow development the workflow developer simply calls this tool and it will automatically generate the required bundle and upload it into the SHIWA Workflow Repository. Such tool was developed for ASKALON, MOTEUR, WS-PGRADE and Triana workflow systems. These examples provide a best practice description for the developers of other workflow systems how to develop a similar tool for their own workflow system.

The second option requires the workflow developer to study the bundle and to manually create and upload it into the repository. This is a quite difficult process and hence usually workflow developers do not use it.

Once the workflow bundle is published in the repository, other workflow developers and end-user scientists can search for it and use it via the SHIWA Portal.

C. SHIWA Portal

The SHIWA portal is a generic purpose WS-PGRADE/gUSE portal that is adapted for the SHIWA system as follows:

1. It is directly connected to the SHIWA Workflow Repository and enables the search inside the repository.
2. Once candidate workflows are found they can be selected and put into a pre-selection list.
3. When the WS-PGRADE workflow is configured any node of the workflow can be configured to take a reusable workflow from the pre-selection list and create a so-called embedded workflow node. (Workflows containing embedded workflow nodes are called meta-workflows [22] since they integrate many different workflows into a single workflow.) If a WS-PGRADE workflow node is associated with a reusable workflow (e.g. a MOTEUR workflow) from the pre-selection list, the execution of this WS-PGRADE workflow node means to invoke the (MOTEUR) workflow enactor with the associated (MOTEUR) workflow.
4. The WS-PGRADE workflow manager is extended with the SHIWA Submission Service [21] which has the capability of submitting the associated reusable workflow to the corresponding workflow enactor that works in the DCI where the reusable workflow was developed. Of course, the user should give the required authentication information before the WS-PGRADE workflow system submits the reusable workflow to the DCI specified in the bundle of the reusable workflow.

D. SHIWA Virtual Organization

The SHIWA Virtual Organization contained two types of grid (gLite and Globus) computing and storage sites from four countries. gLite sites were needed since MOTEUR was developed for gLite and Globus was needed because ASKALON was designed for Globus. WS-PGRADE workflows can run both in gLite and Globus (and even in ARC, UNICORE and BOINC). The SHIWA Virtual Organization contained a VOMS and a myProxy server and a pre-deployed MOTEUR and ASKALON workflow engine. These are used to execute the reusable MOTEUR and ASKALON workflows.

E. SHIWA Workflow Interoperability Practice in ER-Flow

As mentioned in the Introduction, the SHIWA workflow interoperability concept was actively used by four user communities in the ER-FLOW project [9]. The Helio-physics community integrated Taverna workflows and WS-PGRADE workflows. The BioMedical community combined MOTEUR and WS-PGRADE workflows, while the astrophysics community intended to integrate Kepler and WS-PGRADE workflows. Finally, the chemistry community used UNICORE and WS-PGRADE workflows [33].

The experience of these communities was that the black box concept of SHIWA could be used in everyday practice (they uploaded more than 200 workflows into the SHIWA Workflow Repository) but in order to port them from the

SHIWA VO to other VOs some new approach was needed. This new approach is based on the enabling of dynamic DCI deployment possibility into clouds during workflow execution as mentioned in the Introduction and described in detail in the next sections.

III. Infrastructure Aware Scientific Workflows

In order to solve the problem of creating meta-workflows where embedded workflows can run in any kind of DCIs, we proposeto extend the original meta-workflow with nodes responsible for setting up the required infrastructure in clouds. The aim of this workflow extension is to broaden workflow usability, shareability and interoperability. This vision perfectly fits in the previously introduced SHIWA black box concept where reusable workflows are embedded into the actual workflow. With this new concept we can also insert deployment nodes responsible for deploying in a cloud the infrastructure required for executing the embedded reusable workflow. In such workflows besides the traditional computation nodes, infrastructure managementnodes (DEPLOY, UNDEPLOY)can also be placed. For the sake of simplicity in this paper we deal only with DAG (Directed Acyclic Graph) based workflows where arcs represent the data dependency (and data transfer) among the computational nodes and a node can be executed when all the input arcs hold the required data. This semantics can naturally be extended with the new infrastructure management nodes. In fact, it is not obvious how infrastructure management nodes can fit into a scientific workflow and there are many possibilities to define the semantics of such an extended workflow. The most important questions that can be raised on these new nodes are the following:

1. Where to place the DEPLOY and UNDEPLOY nodes in the workflow graph?
2. What is the expected lifetime (the so called scope) of a VI created with a DEPLOY node?
3. How the scope is defined?
4. Should DEPLOY and UNDEPLOY nodes be used in a structured way or can they be placed anywhere without any restriction?
5. Who, when and with which parameters starts these nodes?

Depending on the responses given to these questions there are many options to place these nodes into a workflow graph. To thoroughly discuss all the different possible options is beyond the scope and ambitions of this paper. Here we give only few alternative options that can easily be adopted for the SHIWA environment described in Section II.For the sake of simplicity, we assume that the host machine where the WFM runs is connected to a certain cloud and the DEPLOY nodes create the required VIs in this cloud. This assumption does not influence the placement strategy of DEPLOY and UNDEPLOY nodes but simplifies the number of parameters to be passed them.

In order to show the possible options and their execution mechanisms let's consider a simple but not trivial workflow example (Fig. 1) where the major problems of placing DEPLOY and UNDEPLOY nodes can be highlighted.This workflow consists of 5 nodes. (Notice thatwe use the WS-PGRADE workflow notation during this paper but in principle any other DAG based workflow language could be used as example.) The Autodock node embeds a ready-to-use Autodock workflow that is stored in the SHIWA Repository and requires the usage of a particular BOINC infrastructure. App2 and App3 another two nodes that can run in parallel with Autodock, and they require exactly the same BOINC infrastructure to run. Finally, SEQ1 does some data preparation activities and SEQ2 processes the outputs of the Autodock and App3 nodes. SEQ1 and SEQ2 has no special infrastructure requirements, they can run for example on the same server where the workflow manager runs.

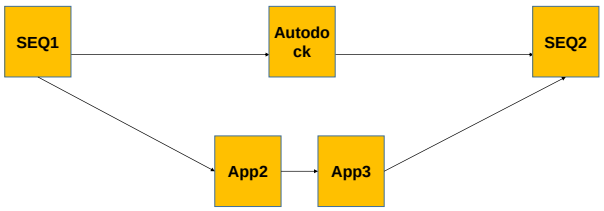

Fig. 1.Workflow example for illustrating the issues of infrastructure aware workflows

The first question is where to place the infrastructure management nodes if the scope of DEPLOY is 1 node. Figure 2 shows a possible placement scheme of this kind of DEPLOY nodes. The idea is that the three nodes requiring the new VI in the cloud (BOINC in the example) are preceded by a DEPLOY node.All the DEPLOY nodes get as parameter the TOSCA descriptor of the BOINC VI and once they are activated by the WFM they build the required BOINC VI in the cloud. Once theirtarget workflow node (pointed by their output arc) is finished the corresponding undeploy functionality is executed automatically by the WFM and hence the user does not have to place any UNDEPLOY node into the workflow graph. The placement scheme of DEPLOY is very simple but there is a significant problem with this concept. The three DEPLOY nodes create redundantly three BOINC infrastructures although the same BOINC VI could be used by Autodock, App2 and App3.

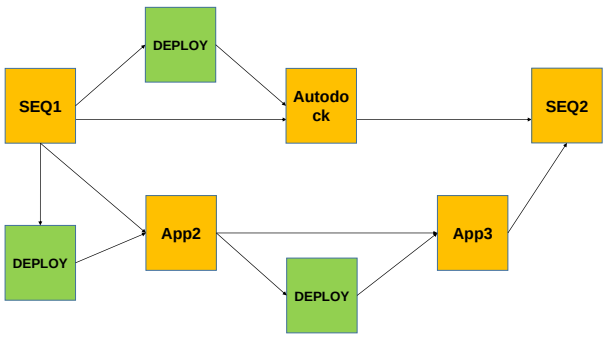

Fig. 2.Workflow example for illustrating the placement of 1-node scope DEPLOY nodes

A possible remedy for this problem is to allow the use of DEPLOY nodes with N-node scope. In this case the scope of the DEPLOY node could be defined by the number of output arcs. Every output arc of the DEPLOY node would be connected to a compute node requiringthe same VI and hence works inside the scope of the DEPLOY node (see Fig. 3). The execution of a pointed compute node inside the scope region

could be executed when both its data input arcs and the input arc coming from the preceding DEPLOY node contain the required information. This information in the case of a DEPLOY output arc is the identifier and access point of the deployed VI. Notice that the compute nodes using the VI should be connected toa scope-closing UNDEPLOY node. Each compute node after finishing its work sends a VI deactivation signal to the scope-closing UNDEPLOY node. The VI deactivation information identifies the VI that should be removed by the UNDEPLOY node. The WFM activates the UNDEPLOY node when all its input arcs contains the VI deactivation signal. This is completely in line with the original node activation semantics of DAG workflows. Notice that this solution eliminates the problem of the 1-node scope scheme since it will create a single VI for all the three compute nodes that need the same VI. The price for this efficiency is that the workflow graph can contain too many arcs making difficult to understand the workflow graph structure.

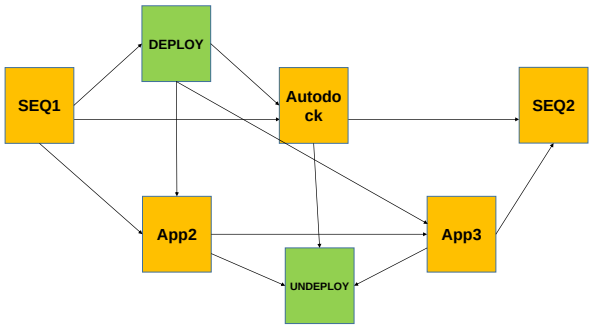

Fig. 3.Workflow example for illustrating the placement of N-node scope DEPLOY nodes

One possible solution for this problem is if those nodes that require the same VI are clustered and placed inside an embedded workflow as shown in Figure 4. In such case a single DEPLOY node is enough to precede the meta-node (containing the clustered sub-graph) and hence this could be a 1-node scope DEPLOY which means that no UNDEPLOY node is needed. It is important to emphasize that such a clustering is not always possible (for example different VIs are required, or node dependencies would make it inefficient) and hence we still need N-node scope DEPLOY nodes and as a consequence UNDEPLOY nodes when node clustering is not possible.

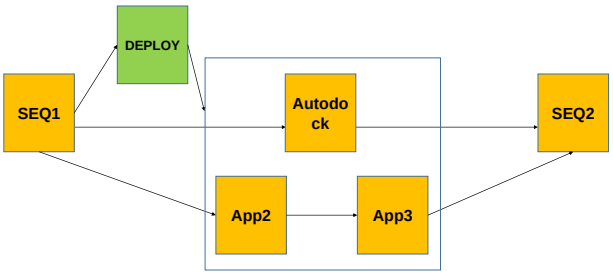

Fig. 4.Workflow example for illustrating the placement of N-node scope DEPLOY nodes

The discussion above gave answers for the first three questions. The fourth question is about the structured way of using DEPLOY and UNDEPLOY nodes. We can see that the solutions discussed above always gave a structured solution of using DEPLOY and UNDEPLOY node. In the case of 1-node scope DEPLOY nodes implicitly we use a structured scheme where the pair of the 1-node scope DEPLOY node is always realized by an implicit UNDEPLOY node that is implemented by the WFM. In the case of N-node scope DEPLOY nodes they always should be used together with a symmetric UNDEPLOY node whose existence can easily be checked at workflow definition time before the workflow execution is started. The simple structuring rule is that if a compute node has an input arc originated from a DEPLOY node, then it should have an output arc that is targeted to the UNDEPLOY node that is the structured pair of the DEPLOY node. This is a useful feature that makes manageable the creation of infrastructure aware workflows. Notice that the unstructured use of DEPLOY and UNDEPLOY nodes could cause many errors that would turn out only at run time and would make the development of infrastructure aware workflows very difficult.

The fifth question is very much related to the implementation issues and hence it will be answered in Section IV.

IV. Implementation of Infrastructure Aware Scientific Workflows

As described in the previous section the introduction of the two infrastructure deployment node types (DEPLOY and UNDEPLOY) enables to solve the problem of integrating various workflows taken from the SHIWA Workflow Repository into a single but complex workflow that can guarantee that the workflow taken from the SHIWA Repository will be executed in the infrastructure specified for it.

In this section we show how to implement the DEPLOY and UNDEPLOY nodes in the WS-PGRADE/gUSE gateway framework which is the portal of the SHIWA Simulation Platform (SSP). It was already described in [22] how the workflows taken from the SHIWA Repository are managed and executed as embedded nodes in WS-PGRADE workflows within the SHIWA Simulation Platform (SSP). Here we focus on how to create the required infrastructure in the cloud before the embedded workflow node is actually invoked by the WS-PGRADE workflow enactor.

The key tool to implement the DEPLOY and UNDEPLOY node types is a TOSCA compliant cloud orchestrator service. Such a service called Occopus is under development in SZTAKI, and its preliminary version was discussed in a previous paper [13] (referred as One Click Cloud Orchestrator). In that work it was introduced as a technique to instantiate and maintain flexible and resilient virtual infrastructures. Since then, Occopus has been released as open source software under Apache v2 license [26]. Before we describe how Occopus relates to the meta-workflows of the WS-PGRADE/gUSE gateway framework, we give a short overview of the Occopus components that allow the prompt creation and fault tolerant maintenance of the virtual infrastructures capable of hosting the various workflow activities. In order to achieve its goals, Occopus uses two kinds of components: ones that are oriented on the support of end users, and ones that focus on the definition, inspection of constantly developing virtual infrastructures.

First, let us focus on the end user oriented components: the template store and the notification service. With the template store, Occopus allows virtual infrastructure designers to create such infrastructure descriptions that on one hand are easily

customizable by end-users and on the other hand are capable to describe the peculiarities of virtual infrastructures. With the exception of the customization options, we will not discuss further the details of the virtual infrastructure description as it is out of scope. The customization options are specified in the description as hints attached to the attributes that the user should be able to change. These hints allow the automated construction of the UI for infrastructure customization (this UI will be further discussed in Section IV/A). The other user oriented component of Occopus is the notification service, that plays crucial role after the infrastructure is customized by the user and its creation is requested from the virtual infrastructure management related components (see in Section IV/A). The notification service enables automated reactions to particular infrastructure maintenance related activities (e.g., when the infrastructure first becomes available or when it has scaled to allow more throughput). This service plays a crucial role in the integration of infrastructure management to workflows as it allows determining when a particular virtual infrastructure becomes available for future workflow activities.

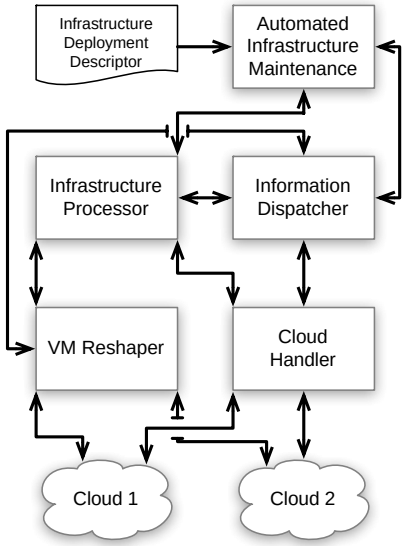

Fig. 5.Occopus virtual infrastructure management components

Next, we turn our attention towards the components actually managing the virtual infrastructures for the end-user (see Fig. 5): *(i)* Automated Infrastructure Maintenance, *(ii)* Infrastructure Processor, *(iii)* Cloud Handler, *(iv)* VM Reshaper, and *(v)* Information Dispatcher. We have defined the Automated Infrastructure Maintenance component as the one responsible to understand the customized deployment descriptors. But this component does not only provide the descriptor processing capabilities but it also offers dependency resolution (so the nodes of the particular instantiated infrastructures are instantiated in a natural order), scalability and error resilience rule evaluation and enactment (so the end user does not have to intervene in its infrastructure's internal operations). The Infrastructure Processor component of Occopus is used to ensure that the definitions of the infrastructure nodes are propagated to the VM Reshaper (which allows runtime reconfiguration of a virtual machine to meet a particular node description). In addition, the Infrastructure Processor sends virtual machine requests to the Cloud Handler that ensures the intended role of the virtual machines after their creation. Next, the Cloud Handler is responsible of selecting a cloud infrastructure that will host a particular virtual machine, and interfacing with the infrastructure provider in a unified manner. Finally, the Information Dispatcher component allows the Automated Infrastructure Maintenance component to determine the current state of the infrastructure to be used during the scaling and error resolution rule evaluation process.

In the coming sub-sections, we will show how these components are exploited when a user defines and executes a meta-workflow in the SHIWA WS-PGRADE/gUSE gateway to allow dynamic virtual infrastructure provisioning for the embedded scientific workflows.

A. The behavior of the infrastructure creation node

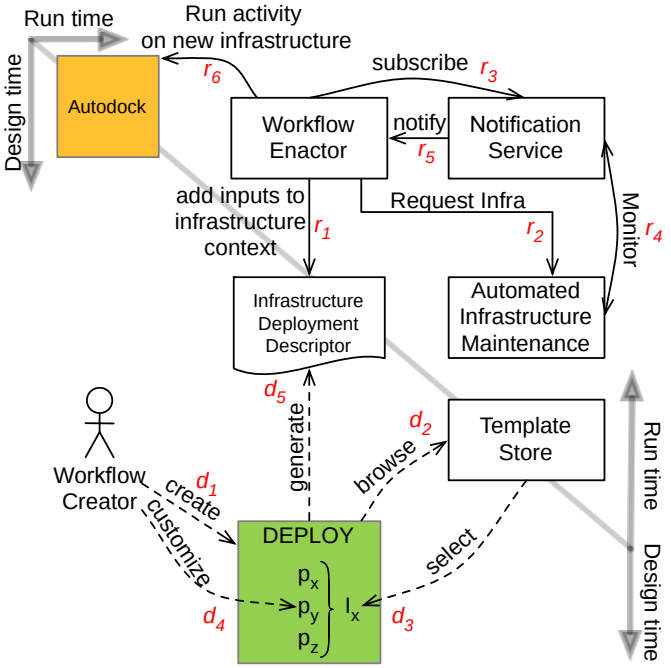

Fig. 6.Definition and behavior of an infrastructure creation node

To allow on demand virtual infrastructure deployments in a workflow environment, we introduced the concept of infrastructure creation nodes (DEPLOY nodes). These nodes are intended to directly interface with the Occopus system but they allow the definition of infrastructure management operations according to customary practices within the workflow ecosystem. In the following, we first will show how workflow creators could incorporate such DEPLOY nodes into their workflows in order to increase their dynamic nature, and then we will show how a workflow enactor can handle these DEPLOY nodes.

Whenever a workflow creator inserts an embedded workflow from the SHIWA Repository that needs a specific kind of infrastructure that are not widely accessible to the user community of the newly constructed workflow, he/she can insert a DEPLOY node into the meta-workflow to be executed before such infrastructure would be needed by the embedded workflow's activities. Fig. 6 represents this design time operation with step d_1 (create). Once the node is created, the workflow system should offer the opportunity to select the kind of infrastructure the workflow creator expects to be available for the workflow after the DEPLOY node has been executed. This operation is embodied in steps d_2 and d_3. As an example, in the WS-PGRADE/gUSE portal, these two steps can be achieved similarly to the SHIWA portal's workflow browsing and selection facilities. However, instead of turning to the SHIWA repository, now the workflow editor is expected to

interface with the Occopus template store. After the workflow creator has selected the infrastructure template (I_x) needed for his/her workflow, the workflow editor enters the infrastructure customization mode (see step d_4 in Fig. 6). In this mode, the workflow creator will specify the parameterization (p_x, p_y, p_z) of the infrastructure I_x (e.g., when a BOINC infrastructure is selected, then one will be able to customize the initial number of worker nodes to be created alongside the BOINC server in the cloud). Finally, when the workflow is saved after the customization, the workflow editor will generate a custom infrastructure deployment descriptor allowing the workflow enactor to use this descriptor during the workflow's runtime. This final step is represented in step d_5.

Next, let us turn our attention to the runtime of the newly edited workflow with a DEPLOY node in it. The top right part of Fig. 6 shows this situation. The figure starts with the workflow enactor's realization that the next node to be executed in the workflow is going to be a DEPLOY node. In such case, first, the enactor will check whether the workflow activities planned to be executed on the new infrastructure need some input files. In such case, as seen in step r_1, the previously generated deployment descriptor is extended to let Occopus know that the input files should be placed in the infrastructure using the particular cloud's virtual machine contextualization technique. Of course, this step can be omitted if there is no input needed, or if the upcoming activities are collecting their input data on alternative ways. With the now completely prepared deployment descriptor, the enactor will request Occopus's Automated Infrastructure Maintenance component to construct the described infrastructure for the workflow (this operation is shown as step r_2 in the figure). The request returns with the future infrastructure's Occopus identifier. This identifier is used in the following subscription step – r_3 – in which the enactor requests Occopus to let it know when the requested infrastructure is constructed according to customizations, definitions and rules found in the deployment descriptor. Inside, Occopus will start monitoring (see step r_4) the infrastructure creation process to ensure the timely notification of the workflow enactor in step r_5. In the final runtime steps, the enactor associates the just created infrastructure to those activities that the workflow creator designated for this specific kind of infrastructure. Then in step r_6, the activities (like the Autodock activity shown in the figure) can start their execution on the new infrastructure just as if the infrastructure has been there statically.

B. The behavior of the infrastructure destruction node

Naturally, if infrastructures can be created dynamically during workflow runtime, their destruction should be also managed dynamically. To this end, we propose the concept of infrastructure destruction nodes (UNDEPLOY nodes). Similarly to the infrastructure creation node, these nodes couldbe also placed into the workflow (but as discussed before it is only necessary to be placed as structured pairs of N-node scope DEPLOY nodes). Compared to the complex definition of the creation nodes, destruction nodes only need to refer to the infrastructure no longer desired. This information is delivered in the deactivation data on the input arc of UNDEPLOY.When the deactivation data arrived on all input arcs of the UNDEPLOY node the workflow manager sends the destruction request to Occopus with the identifier of the VI to be removed from the cloud.

V. INFRASTRUCTURE AWARE WORKFLOW MANAGERS

In Section III we have discussed the possibilities of explicitly using DEPLOY and UNDEPLOY nodes in scientific workflows and we called these workflows as infrastructure aware workflows. These workflows enable users to initiate the creation of a VI at any place of their workflow. Though the proposed concept is structured and relatively simple to use the ideal solution would be to make the creation of the required VIs completely transparent for the users. In this section we investigate how to organize workflows, workflow repositories and workflow managers to make this concept feasible.

Since there is no need for DEPLOY and UNDEPLOY nodes in this concept rather the workflow managers should be able to recognize when a new VI should be deployed or a previously created VI should be removed there is no language aspect of this concept. We need a more intelligent WFM than used so far and we will call this new type of more intelligent WFMs as infrastructure aware workflow managers.

For a typical state-of-the-art WFM two kinds of information is needed for every WF node in order to submit the corresponding job into a concrete infrastructure:

<INF_type, concrete_INF_id>

Based on this information the WFM can create a JSDL according to the BES standards [27] and can send the job generated from the workflow node to the job submission service together with this JSDL. The job submission service tries to submit the job to the identified concrete infrastructure (for example to GT5). If the submission fails then current WFMs send back an error report to the user and are unable to continue the execution of the workflow. Since for example grid infrastructures are quite unreliable this problem frequently happens.

The infrastructure aware WFM is able to overcome such a problematic situation. If it encounters a job submission problem it can automatically initiate deployment of the concrete infrastructure as a VI in a cloud and once the VI is created it can submit the job to the newly created VI. Obviously, this approach requires that the TOSCA descriptor of the required concrete infrastructure and a cloud orchestrator service that can process the TOSCA descriptor should be available for the WFM. Based on the available TOSCA descriptor the WFM and the TOSCA compliant cloud orchestrator can manage the creation and usage of the required VI.Once the job submission and execution successfullycompleted in the VI the WFM can invoke again the cloud orchestrator to remove the VI from the cloud.

Notice that the described mechanism implements a1-node scope DEPLOY node and hence the usage of explicitly placed 1-node scope DEPLOY nodes is not needed in this concept.

The next question is how to substitute the mechanism of N-node scope DEPLOY nodes and their structured pair UNDEPLOY node. The solution is very simple. After successfully finishing a job submission and execution in a VI the WFM does not request immediately the cloud orchestrator to remove the VI from the cloud. Rather for every concrete infrastructure that is used in the workflow the WFM creates a Deploy Table that contains the following information:

<concrete_INF_id,TOSCA_Descriptor,VI_access,job_number,VI_exists, time_to_remove>

The meaning of the items in this record is as follows:

- concrete_INF_id: identifies a concrete infrastructure used in the workflow
- TOSCA_Descriptor: identifies the TOSCA descriptor of the concrete infrastructure
- VI_access: information record needed to access the VI in the cloud (initial value is EMPTY)
- job_number: shows the number of jobs actively using the VI generated for this concrete infrastructure (initial value is 0)
- VI_exists: a Boolean value indicating if the VI exists in the cloud (initial value is FALSE)
- time_to_remove: a timer, if it reaches a configurable predefined value, the VI should be removed from the cloud (initial value is 0)

When a concrete infrastructure does not respond correctly for job submission the WFM checks if the corresponding VI already exists using the VI_exists boolean value. If the VI exists then the WFM updates the concrete infrastructure access information in the JSDL (using the stored VI_access information) and the job submission service can submit the job to the VI based on this access information. The WFM also increments the job_number counter. If the value time_to_remove is not zero, it sets this value to zero (see the explanation later).

If the VI does not exist, then WFM calls the TOSCA compliant cloud orchestrator service with the TOSCA_Descriptor. Once the VI is deployed the cloud orchestrator gives back the VI access information that is written into VI_access information by the WFM. Then the WFM updates the boolean value VI_exists to TRUE, increments the job_number counter and updates the job JSDL based on the VI access information and send the new JSDL to the job submission service.

Once the job submission and execution is successfully finished in the VI, the WFM checks the DEPLOY table record. First decrements the job_number counter and then checks its value. If it is more than 0 no more action is required. The VI should still work since other jobs use it. If the value job_number is 0, then the process of removing the VI can be initiated. Could be done immediately but that would be not optimal since it can happen that very quickly another node of the workflow would require the same VI and then it would be very costly to first remove the VI and then again deploy it. Therefore before removing a VI it is better to wait a certain amount of time (at least the required time of creating and removing this VI). There could be many options to set up this time threshold. The simplest one is if the WFM owner sets up this time threshold as a parameter of the WFM. (More complex and fine-tuned solutions can be applied but these techniques do not influence the generic work of the infrastructure aware WFM and hence we use in this paper this simplest mechanism). According to the simplest option the initiation of removing the VI means that the WFM sets the value time_to_remove to the threshold given by the WFM owner. Once the time_to_remove value reaches 0 the WFM calls the cloud orchestration service to remove VI from the cloud.

Notice that the described mechanism automatically recognizes when a DEPLOY node would be needed in the graph and also automatically recognizes when the closing UNDEPLOY node would be required. Therefore the user does not have to deal with the placement and configuration of DELOY and UNDEPLOY nodes.

In our experimental system the WFM is WS-PGRADE/gUSE and the BES-compliant job submission service is DCI Bridge [28]. The TOSCA-compliant cloud orchestrator is Occopus. Currently we have no repository where the TOSCA descriptors of the concrete infrastructures could be stored. In ideal case the SHIWA Workflow Repository would be this place storing a simplified Deploy table with the following records:

<concrete_INF_id,TOSCA_Descriptor>

The advantage storing this Deploy table in the SHIWA Workflow Repository is to store every workflow related information in the same repository. When a user community uses a certain concrete infrastructure and develops workflows for such concrete infrastructure it is enough to create and store once the required TOSCA_Descriptor for the substituting VI. No matter how many workflows they develop and store in the SHIWA Repository all these workflows can refer this common TOSCA descriptor in the Deploy Table. Of course, if the concrete infrastructure is changed then version numbering is important and for every new version of the concrete infrastructure a new TOSCA descriptor should be created and stored in the Deploy table. The SHIWA Repository already support versioning of the concrete infrastructures so this versioning could be easily used in the Deploy table, too.

VI. Related Work

The question of workflow interoperability has been investigated in depth, see e.g. [8] where levels of possible interoperation are specified. From a practical point of view (simplifying [8]), approaches for workflow interoperability can be divided into ones centered around workflow languages and graph based ones. Language based interoperability solutions tackle with translating from one workflow description language to another or provide an intermediate common language [18]. This is a fine grained solution and the “white box” approach in this paper corresponds to this option. Graph based approaches try to embed graphs into one another [22][17] representing a coarse grained concept and corresponds to the black box approach presented in the paper. The work presented in this paper is aimed at supporting the latter, graph based (black box) type of interoperability solved by dynamic resource orchestration. Dynamic resource orchestration – similar to the Occopus concept presented in this paper – for on-demand resource provisioning is a known technique, even in the context of workflows nevertheless, to our best knowledge, not for achieving interoperability. These resource orchestration concepts are introduced in the followings.

One type of orchestration tools, such as Saltstack [20], Puppet [19], Chef [1], Docker [5], Juju [23] and Cloudify [4], covers the development and operations aspectsand are aimed at automating development and system administration tasks such as delivery, testing and maintenance to improve reliability, security and so on. These can be considered as static ones in comparison with Occopus presented in this paper. Beyond these general-purpose utilities other orchestration tools are aimed at specific goals, typically on-the-fly resource provisioning, adaptation, load distribution, QoS and maintaining SLA – but none of them is tied to workflow interoperability.

Orchestrator [12] is aimed at resource provisioning in sensor-rich mobile platforms where it enables multiple simultaneous, context aware applications sharing highly scarce and dynamic resources. Applications submit high-level context specifications and comply with Orchestrator's resource allocation. Resource selection and binding is postponed until resources' availability is sufficiently explored.

Similarly, Merwe at al. define a Cloud Control Architecture for a ubiquitous cloud computing infrastructure [16]where orchestration is realized as a separate layer and interconnects the Service Abstraction (presents service logic to the users) and Intelligence (gathers information about the cloud infrastructure) and derives abstract knowledge. The Orchestration layer collects both the requests from Service Abstraction and actual data from Intelligence and makes decision about initial placements, resource allocation, resource adjustment and movement of resources.

Lorincz et al. present a very different way or resource orchestration in Pixie: resource tickets [14]. A ticket is an abstraction for a certain part (capacity) of a resource and all orchestration actions are mediated via the tickets. Tickets are generated by resource allocators and managed by resource brokers. A ticket provides information about the resource, the allocated capacity and the timeframe. Resources can be manipulated by operations on tickets such as join (increasing resource capacity), split (sharing), revoke or redeem (collecting specific tickets for a certain operation) just to mention a few. This approach also decouples actual resources from resource requests and gives a great flexibility in planning, advance requests and adaptation.

Huang et al. [10] introduce an adaptive resource orchestration scheme in order to find a balance between performance, cost and energy consumption while maintaining SLAs. Their technique aims at a sub-optimal allocation scheme by predicting resource needs and planning resource re-distribution. Each time a VM is demanded, a prediction may be launched to evaluate future needs and possibly, a new allocation is planned in order to optimize the efficiency of global deployment.

Following this trend, Maurer et al. [15] investigate adaptive resource configuration from a SLA/QoS point of view according to the well known monitoring-analysis-planning-execution (MAPE) cycle of adaptive systems. The possible actions to fine tune the performance (and other parameters) of a VM are categorized hierarchically as so called escalation levels and asserted in a knowledge base. Generally, much finer grained actions (e.g. memory configuration) are considered than in our work.

Chen et al. [2] present Sulcata, an on-line virtual cluster provisioning. They assume a pool of various physical resources that act as VM containers. Upon a user's request the system deploys a virtual cluster on-the-fly involving VM image preparation, VM creation and configuration and VM reboot. They clearly solve basic functionalities of dynamic infrastructure provisioning; furthermore, the solution largely focuses on minimizing the overhead of VM image handling and deployment and proposes a resource mapping scheme.

Dörnemann et al. [7] address the issue of on-the-fly infrastructure provisioning for workflow applications. In an earlier work [6] they presented an on demand deployment scheme to avoid peak loads but the solution showed some shortcomings for workflows with respect to throughput and cost. To find a trade-off between task based scheduling (imprecise) and graph based scheduling (complex) of workflows, [7] proposes a critical path based scheduling. The graph is annotated with information on anticipated run times and data transfers to calculate the makespan. Possible allocations for critical paths are predicted by genetic algorithms and the process is iterated until a mapping with minimal estimated runtime is reached.

Vukojevic et al. [24] present a very similar solution to ours in a service oriented scenario to support simulation workflows. Their aim is to provide and redeem services on-demand according to the progress of the workflow. The core of the solution is dynamic binding with software stack provisioning. Albeit the functionality is close to ours, the technical realization is entirely different due to the service oriented approach and ultimately, the solution is not aimed at supporting interoperability.

A special case of the infrastructure aware workflow concept has been implemented in University of Westminster [30]. They have created three scripts: a Hadoop deployment script, a Hadoop execution script and a Hadoop destroy script. In their approach these scripts can be used as independent workflow nodes called Deploy, Execute and Destroy, respectively. These nodes can be placed anywhere in a workflow and as a result, this workflow can be considered as a special infrastructure aware workflow where the placement of Deploy and Destroy nodes is unstructured and they support only the creation and usage of Hadoop VIs.

VI. Conclusions

The extension of workflows with DEPLOY and UNDEPLOY nodes enables workflow developers to create workflows that can dynamically build the infrastructures they need to run on. We showed that this new concept of infrastructure aware workflows solves the problem of the SHIWA black box workflow interoperability concept. Moreover, this new concept will open a new horizon to create extremely dynamic and flexible workflows that can easily adapt themselves to the infrastructure where they run and if the underlying infrastructure does not fit them they can easily customize the infrastructure on-the-fly.

As was shown in Section III and V, infrastructureaware workflows can be created in two ways: (1) as user defined when the user defines and places the DEPLOY nodes or (2) automatic when the WFM service decides when and which infrastructure is to be used and hence we call this concept as infrastructure aware workflow managers Both concepts have advantages and drawbacks. The infrastructure aware workflow concept enables users to directly tailor workflows to certain VIs that already have got TOSCA-compliant descriptors. For example, using Occopus we have already created BOINC infrastructure descriptors where the number of BOINC clients can be parameterized. Therefore DEPLOY nodes that require BOINC VI descriptors can already be used and workflows running on BOINC infrastructures (and stored in the SHIWA Repository) can be placed in meta-workflows even if the workflow manager service is not connected to any existing BOINC system but at least to a cloud system.

The infrastructure aware workflow manager concept is a very generic solution for running workflows even if their original concrete infrastructure is not available. This solution is

perfectly transparent to the users. However, it requires that the concrete infrastructures used in a workflow have their TOSCA-compliant VI descriptors. Currently, it is not typical that user communities create the required VI descriptors. However, if in the future workflow managers are extended to be able to handle VI descriptors as proposed in this paper many user communities can find useful to put the effort and create the required VI descriptors since it has many benefits for them:

- Workflows extended with VI descriptors of the used concrete infrastructures can run on a very reliable way even on non-reliable infrastructures.
- If a concrete infrastructure is not supported anymore the existing workflows designed to run on this infrastructure still can be used without changing the workflows or porting them to new infrastructures.

We also showed a tool (Occopus) that enables the implementation of this new workflow creation concept. The integration of Occopus and the WS-PGRADE/gUSE gateway framework provide a full-fledged implementation of the infrastructure-aware workflow concept and hence any community interested in using the concept can get this implementation. Certainly, the user communities of the SHIWA Simulation Platform are good candidates to use this technology but we can envisage other user communities with similar requirements of building dynamically their required infrastructure.

Both Occopus and WS-PGRADE/gUSE are already released open source software systems. WS-PGRADE/gUSE and the SHIWA Simulation Platform are used in production by 20-30 user communities who constantly give feedback on the usability of these technologies. The integration work of Occopus and WS-PGRADE/gUSE just started but it does not raise more issues than the previous integration of WS-PGRADE/gUSE with various cloud systems either directly or via the CloudBroker Platform [31]. Since WS-PGRADE/gUSE has already been integrated with clouds once the DEPLOY node built up the required infrastructure in a cloud via Occopus the WS-PGRADE workflow enactor can submit those workflow nodes to the target cloud that require the dynamically built infrastructure to run.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 283481 (SCI-BUS), no. 312579 (ER-FLOW) and no. 608886 (CloudSME). This paper is a significantly revised and extended version of the workshop paper [32].

REFERENCES

[1]Chef.http://www.getchef.com, 2014.

[2]Yang Chen, Tianyu Wo, and Jianxin Li. An efficient resource management system for on-line virtual clusterprovision. In IEEE CLOUD, pages 72-79, 2009.

[3] CloudBroker Platform. http://cloudbroker.com/, 2013.

[4]Cloudify.http://www.cloudifysource.org, 2014.

[5]Docker.https://www.docker.io, 2014.

[6]Tim Dörnemann, Ernst Juhnke, and Bernd Freisleben.On-demand resource provisioning for bpel workflows using amazon's elastic compute cloud. In IEEE/ACM Int. Symp. on Cluster Computing and the Grid(CCGrid), pages 140-147, 2009.

[7]Tim Dörnemann, Ernst Juhnke, Thomas Noll, Dominik Seiler, and BerndFreisleben.Data flow driven scheduling of bpel workflows using cloud resources. In IEEE CLOUD, pp. 196-203, 2010.

[8]Erik Elmroth, Francisco Hernández, Johan Tordsson: Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Gen. Comp. Syst. 26(2): 245-256 (2010)

[9] Building a European Research Community through Interoperable Workflows and Data (ER-flow) Eu FP7 project. Online: http://www.erflow.eu/, 2013.

[10]Chenn-Jung Huang, Chih-Tai Guan, Heng-Ming Chen, Yu-Wu Wang, Shun-ChihChang,Ching-Yu Li, and Chuan-Hsiang Weng. An adaptive resource management scheme in cloud computing.Eng. Appl. of AI, 26(1):382-389, 2013.

[11] P Kacsuk, Z Farkas, M Kozlovszky, G Hermann, A Balasko, K KaroczkaiWS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities, Journal of Grid Computing 10 (4), 601-630, 2012

[12]S. Kangetal. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In IEEE International Conference on Pervasive Computing and Communications, 2010.

[13] Gabor Kecskemeti, Mark Gergely, Adam Visegradi, Zsolt Nemeth, Jozsef Kovacs, Peter Kacsuk: One Click Cloud Orchestrator: bringing Complex Applications Effortlessly to the Clouds. In the 2nd InternationalDIHCWorkshop held in conjunction with Euro-Par, Porto, Portugal, August, 2014.

[14]K.Lorincz, B.r.Chen, J.Waterman, G.Werner-Allen, and M.Welsh. Resource aware programming in the pixie os. In SenSys, pages 211-224, 2008.

[15] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Adaptive resource configuration for cloud infrastructure management. Future Generation Comp. Syst., 29(2):472-487, 2013.

[16] J.Van der Merwe, K.Ramakrishnan, M.Fairchild, A.Flavel, J.Houle, H.A. Lagar-Cavilla, and J.Mulligan. Towards a ubiquitous cloud computing infrastructure. In Proceedings of the IEEE LANMAN Workshop, 2010.

[17]Beth Plale, EranChinthaka Withana, Chathura Herath, Kavitha Chandrasekar, Yuan Luo: Effectiveness of Hybrid Workflow Systems for Computational Science. ICCS 2012: 508-517

[18]Plankensteiner, Kassian, et al. "Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures." Journal of grid computing 11.3 (2013): 429-455.

[19]Puppet.http://puppetlabs.com, 2014.

[20]SaltStack.http://www.saltstack.com, 2014.

[21] SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs (SHIWA) Eu FP7 project. Online: http://www.shiwa-workflow.eu/, 2013.

[22]Gabor Terstyanszky, Tamas Kukla, Tamas Kiss, Peter Kacsuk, Akos Balasko, Zoltan Farkas: Enabling scientific workflow sharing through coarse-grained interoperability, Future Generation Computer Systems, Volume 37, July 2014, Pages 46-59

[23]Ubuntu Juju: http://juju.ubuntu.com, accessed in Dec. 2014.

[24]Karolina Vukojevic-Haupt, Dimka Karastoyanova, Frank Leymann: On-demand Provisioning of Infrastructure, Middleware and Services for Simulation Workflows. SOCA 2013: 91-98

[25]TOSCA: https://en.wikipedia.org/wiki/OASIS_TOSCA, accessed in Dec. 2015.

[26] Occopus: http://occopus.lpds.sztaki.hu/, accessed in Dec. 2015.

[27] OGF BES standard: http://grid.pd.infn.it/NA5/bes-wg.html, accessed in Dec. 2015.

[28] Miklos Kozlovszky, Krisztián Karóczkai, István Márton, Péter Kacsuk, Tibor Gottdank:DCI Bridge: Executing WS-PGRADE Workflows in Distributed Computing Infrastructures. In Science Gateways for Distributed Computing Infrastructures, Book Chapter, Springer Verlag, 2014.

[29] Zoltán Farkas, Ákos Hajnal, Péter Kacsuk. "WS-PGRADE/gUSE and Clouds". In: Péter Kacsuk (ed.), Science Gateways for Distributed Computing Infrastructures: Development Framework and Exploitation by Scientific User Communities, Springer, 2014. pp. 97-109.

[30] Gugnani S, Kiss, T. Extending Scientific Workflow Systems to Support MapReduce Based Applications in the Cloud, 7th International Workshop on Science Gateways, IWSG 2015, 3-5 June, 2015, Budapest, Hungary, pp 16-21, DOI 10.1109/IWSG.2015.15.

[31] CloudBroker Platform: http://cloudbroker.com/, accessed in Dec. 2015.

[32] Kacsuk P, Kecskemeti G, Kertesz A, Nemeth Z, Visegradi A, Gergely M: Infrastructure Aware Scientific Workflows and Their Support by a Science Gateway. 7th International Workshop on Science Gateways - IWSG 2015: proceedings. 2015. pp. 22-27.

[33] J. Krüger et al., The MoSGrid Science Gateway – A Complete Solution for Molecular Simulations. Journal of Chemical Theory and Computation, 10 (6), pp. 2232-2245, 2014. DOI: 10.1021/ct500159h