Deployable Self-contained Workflow Models

Benjamin Weder(✉), Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Michael Zimmermann

Institute for Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{benjamin.weder,uwe.breitenbuecher,kalman.kepes,frank.leymann, michael.zimmermann}@iaas.uni-stuttgart.de

Abstract. Service composition is a popular approach for building software applications from several individual services. Using imperative workflow technologies, service compositions can be specified as workflow models comprising activities that are implemented, e.g., by service calls or scripts. While scripts are typically included in the workflow model itself and can be executed directly by the workflow engine, the required services must be deployed in a separate step. Moreover, to enable their invocation, an additional step is required to configure the workflow model regarding the endpoints of the deployed services, i.e., IP-address, port, etc. However, a manual deployment of services and configuration of the workflow model are complex, time-consuming, and error-prone tasks. In this paper, we present an approach that enables defining service compositions in a self-contained manner using imperative workflow technology. For this, the workflow models can be packaged with all necessary deployment models and software artifacts that implement the required services. As a result, the service deployment in the target environment where the workflow is executed as well as the configuration of the workflow with the endpoint information of the services can be automated completely. We validate the technical feasibility of our approach by a prototypical implementation based on the TOSCA standard and OpenTOSCA.

Keywords: Service composition · Workflow technology · Service deployment automation · Configuration automation

1 Introduction

A popular approach for building applications by combining several individual services is called *service composition*, which can reduce the time and cost to develop new services or applications significantly [7,13]. Service compositions can be specified using imperative workflow languages, such as the *Business Process Execution Language (BPEL)* [14], to benefit from their robustness and features like automatic recovery [12]. Imperative workflow models usually comprise activities that can be executed in the workflow engine, like script calls, and invocations

A. Brogi et al. (Eds.): ESOCC 2020, LNCS 12054, pp. 85–96, 2020.
https://doi.org/10.1007/978-3-030-44769-4_7

of services that run in the environment. The endpoints of available services can be retrieved using a *service registry*. Then, the services have to be bound to the workflow, which means the workflow is configured with the required information to access the services, such as the used protocols or the service endpoints [13].

However, for the successful binding, the required services must be running and accessible by the workflow [18]. Services that are provided over the internet are usually always on and can be accessed from any place if they are not protected by security mechanisms, like firewalls [8,12]. Thus, in general, the binding is feasible independent of the execution environment, e.g., the network of the workflow engine executing the workflow. However, if a workflow requires a service that is not publicly available, the service binding, and therefore, the workflow execution fails. Hence, the missing services have to be deployed by the user to execute the workflow successfully. However, a manual determination of the services that are required, as well as the deployment of these services, is a complex, time-consuming, and error-prone task and not suited for non-technical users [2]. Additionally, the service binding with the deployed services has to be performed by the user, or the service has to be registered correctly with the service registry. An error in the configuration, like a wrong IP-address, leads to the failure of the overall workflow. Furthermore, if a service is migrated, e.g., to another virtual machine, the workflow configuration and the service registry have to be updated correspondingly. Otherwise, the service is no longer accessible, and the workflow execution fails. Therefore, this process leads to a lot of manual work, which is error-prone and should be automated as far as possible.

Many services are not offered over the internet, and therefore, the user of the workflow is in charge of deploying these services. Furthermore, there may also be technical reasons to deploy a service that is available over the internet close to the workflow engine executing the workflow, e.g., to reduce the latency of the service interactions or the required network bandwidth. An example is a workflow processing big data, which would overload the network if the used services are deployed outside the local environment. Thus, to enable the execution of such workflows, the required services must be deployed in the target environment by the user before the workflow execution. However, this leads to the previously described problems, such as erroneous configurations due to human errors.

In this paper, we tackle these challenges by an approach, which allows packaging imperative workflows with all necessary deployment models to deploy the required services of the workflow as a self-contained archive. It consists of the workflow model and a set of deployment models for the required services, which are attached to the activities of the workflow that invoke the services. Additionally, our approach addresses the automatic deployment of all required services in the target environment. Finally, the approach includes the automatic configuration of the workflow with the endpoint information of the deployed services, and therefore, enables defining imperative workflows in a self-contained manner without additional manual tasks to set up required services in the environment.

2 Fundamentals and Problem Statement

In this section, we introduce fundamentals about service composition approaches, imperative workflow technologies, and the deployment of services. Furthermore, we present the problem statement which underlies our approach.

2.1 Service Composition

The creation of new applications by combining existing services is denoted as *service composition* [11]. Service composition can reduce development time and cost significantly as existing functionality is reused instead of implementing it again. In *static service composition*, the required service functionalities and the order in which they have to be invoked are specified at the design time of the service composition [3]. Furthermore, the concrete service implementation must be selected for each required functionality, and the service composition has to be configured to invoke them, which is referred to as *binding* [13]. The binding includes the configuration of the required protocol to invoke the service, the message format, and the endpoint, i.e., the IP address and port of the service. The available services can be retrieved using a *service registry*, which provides binding information for running services to a requester [9]. Thereby, the selection and binding of services during the development of the service composition is called *static binding*. In contrast, the binding at runtime is referred to as *dynamic binding*, which allows to dynamically select a suited service based on non-functional requirements. In this paper, we focus on the static binding of services.

2.2 Imperative Workflow Technology

Service compositions can be specified using *imperative workflow languages*, such as the *Business Process Execution Language (BPEL)* [14]. An imperative workflow consists of a predefined set of activities that have to be executed to achieve the goals of the workflow [5]. Activities can be divided into different categories, e.g., activities that invoke web services or activities that require an human action. The different activities are connected by *control flow* and *data flow edges* [12]. Control flow edges specify a partial order in which the activities of the workflow have to be executed. In contrast, data flow edges define which parts of the output data of an activity must be transferred to which other activities. Two benefits of using workflow technologies are scalability and robustness [5]. Another advantage is the comprehensive error handling mechanisms implemented in most workflow languages and engines. These mechanisms, e.g., allow executing activities in a transactional manner and role changes back in case of an error. Thus, workflows can be executed robustly and provide high-availability to the user.

Due to these advantages, workflows are essential for the implementation of long-running *business processes* [12]. Such business processes have to be executed (i) reliably, (ii) robustly, (iii) in parallel, and (iv) provide high-availability to the user to achieve the maximum business value. The implementation of programs, e.g., written in programming languages such as Java or C, that fulfill these

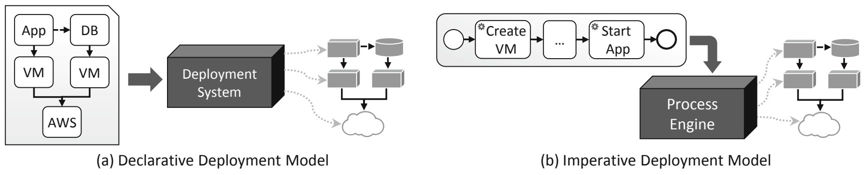

Fig. 1. Deployment model approaches.

properties is a complex and time-consuming task, as they have to be designed specifically with these non-functional properties in mind. In contrast, workflow management systems are general-purpose systems and provide the needed properties directly to the user without the need to implement or adapt them for a certain use case. Hence, they ease the development of workflows implementing business processes [5]. Therefore, workflow technology is of vital importance for the implementation and execution of long-running business processes.

2.3 Service Deployment

For our approach, we distinguish between *provided services*, that are offered by a provider over a network, e.g., the internet, and *self-hosted services*, for which the required software artifacts for the deployment are available, but for which the user is in charge of deploying them. Provided services are *"always-on"*, which means the user can directly use them and is not in charge of creating or deleting them [12]. Examples of this kind of service are Google Maps, Dropbox, or Spotify. In contrast, self-hosted services do not always run on the infrastructure of some provider, and thus, must be deployed by the user before using them in a workflow. All utilized services must be *available* to execute a workflow successfully. This means the services have to respond to requests and return correct results [12]. We focus on self-hosted services in our approach as the user is in charge of deploying them and keeping them available as long as they are needed.

However, the deployment of the required self-hosted services is a complex, time-consuming, and error-prone task [2]. The infrastructure, such as a virtual machine, has to be prepared, the dependencies of the service have to be installed, and the software artifacts of the service have to be transferred to the prepared infrastructure. Furthermore, the service must be configured with the required certificates, and the needed authentication has to be set up. Additionally, the workflow models have to be configured using the endpoint information of the deployed services to access them during runtime [12]. Therefore, a lot of manual work has to be done, and this process should be automated as far as possible.

In recent years several technologies for automating the deployment and management of applications have been developed, such as *Terraform*[1] or *Kubernetes*[2] [20]. Using these technologies, applications are described as reusable

[1] https://terraform.io.
[2] https://kubernetes.io.

deployment models, which can be used to instantiate the application fully automatically. Depending on the modeling approach, deployment models can be divided into two classes, as shown in Fig. 1: *declarative* and *imperative deployment models* [6]. A declarative deployment model describes the structure of an application, including all software and hardware components and their relations. In contrast, imperative deployment models express the deployment process in a procedural manner and contain all activities that have to be executed to deploy the application, as well as the execution order of these activities. Such imperative deployment models can be defined using workflow languages such as BPEL.

Thus, deployment automation technologies can be utilized to deploy required self-hosted services automatically in the target environment. However, the deployment automation technologies are not integrated with workflows and do not update the endpoint information of the activities invoking the services. Hence, the user has to trigger the deployment of the services using a deployment system, retrieve the endpoints from the deployed services, and configure the workflow according to the endpoints. As outlined previously, this process is complex and time-consuming for non-technical users and can lead to configuration errors.

2.4 Problem Statement

As described in the previous subsections, workflow technology is essential for the execution of long-running business processes. However, some of the used services are usually not available over the internet and have to be deployed by the user. Hence, (i) the deployment models for the required self-hosted services have to be determined first. This is complex if there are repositories with lots of deployment models, as it is unclear for non-technical users how to search and select appropriate deployment models. Additionally, (ii) the determined deployment models must be transferred into the target environment for the workflow execution. Further, (iii) required services have to be deployed by passing the corresponding deployment models to a deployment system. Finally, (iv) the services have to be bound to the workflow to access them on runtime. Hence, a lot of complex and time-consuming work has to be performed to prepare the target environment and the workflow for the execution and this process should be automated. Therefore, the resulting research question for this work can be formulated as follows: *"How can business processes be modeled in a self-contained manner and be deployed in the target environment fully automatically including all required services?"*

3 Self-contained Workflow Models

To enable packaging and deploying workflow models that require services that are not provided with the "always-on" property over the network, a self-contained packaging format is needed. Without such a packaging format, the required self-hosted services of a workflow have to be determined manually and deployed

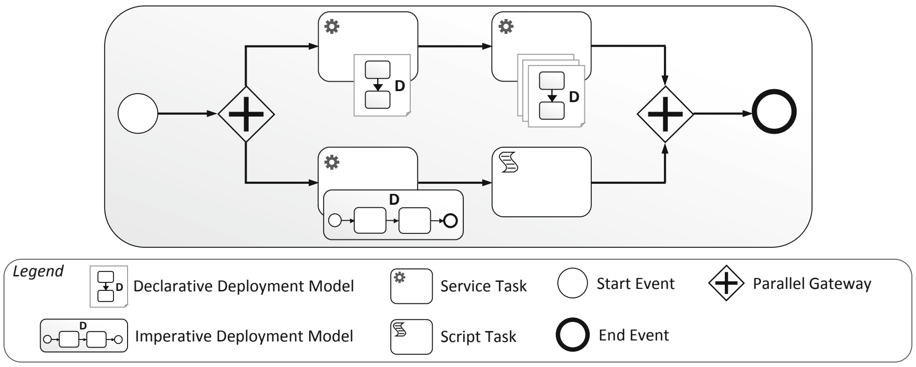

Fig. 2. Our new modeling approach: self-contained imperative workflow models.

in the environment to execute the workflow successfully. Thus, our goal is to develop a packaging format that enables bundling all required information.

The conceptual structure of a *self-contained workflow model* is depicted in Fig. 2. It contains the workflow which can be modeled, e.g., using a standardized workflow language such as BPEL. In the example, the workflow starts, performs two sequences of two activities in parallel, and terminates afterward. However, in contrast to existing workflow archives, deployment models can be added and linked by the activities in the self-contained workflow model. For example, the activity on the top-left references a declarative deployment model. This deployment model can be used to deploy the service that is invoked by the activity. If multiple deployment models for services with the same functionality exist, they can all be linked by the invoking activities (see top-right activity). E.g., one deployment model could deploy the service on a private cloud, while the other could use a local workstation. Further, different deployment models can implement the same service providing various non-functional properties, like response time or security. Hence, a selection based on non-functional requirements of the user or available hardware in the target environment can be performed.

In addition to declarative deployment models, imperative deployment models can be referenced by activities of the self-contained workflow model too (see bottom-left activity). While declarative deployment models simplify common and non-complex application deployments and require only limited technical expertise, imperative deployment models can be modified arbitrarily [6]. Hence, they are better suited for complex deployments with a lot of custom-tailored components. Therefore, our approach allows utilizing both kinds of deployment models to deploy a required service to be generally applicable.

Finally, a self-contained workflow model can also contain activities that do not require the deployment of a service, and thus, have no reference to a service deployment model. For example, an activity that is implemented by a script can be executed within the workflow engine and has no external dependency on a service (see bottom-right activity). Additionally, some activities have to be per-

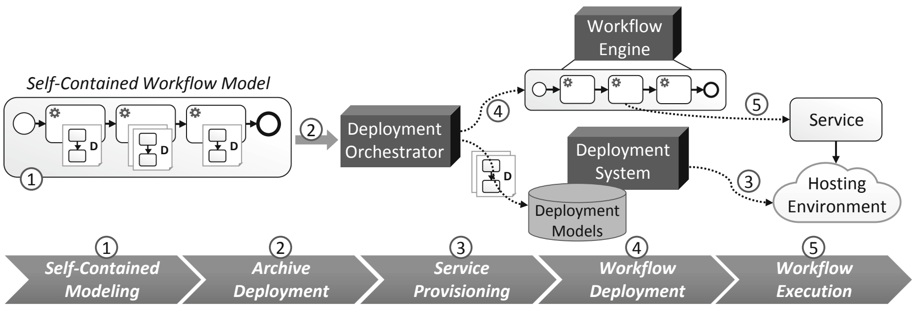

Fig. 3. Overview of the approach for self-contained imperative workflow models.

formed by humans, such as the physical set up of a device. Hence, depending on the required software tools to perform the human task, deployment models may be referenced. Further, activities can invoke provided services that are accessible over the network and do not have to be deployed before the workflow execution.

Self-contained workflow models enable to define imperative workflows implementing service compositions in a self-contained manner with all required service deployment models. Hence, the required services do not have to be determined and transferred into the target environment by the user, which can be a complex task if there are many deployment models available. However, the user is still in charge of deploying the services and configuring the workflow with the endpoints. Therefore, an approach to automate these tasks is required.

4 Automatic Service Deployment

After transferring the self-contained workflow model into the target environment, the required services must be deployed by using the included service deployment models. Furthermore, the endpoints of the deployed services have to be retrieved, and the services have to be bound to the workflow using this endpoint information. In this section, we present an approach to automate these tasks.

Figure 3 gives an overview of our approach. It covers all steps from the definition of the self-contained workflow model to the workflow execution. In the first step, the user models his workflow utilizing a suited modeling tool. The modeling tool presents the available service deployment models to the user, and therefore, allows referencing them within activities of the workflow. After finishing the modeling, the workflow is packaged as a self-contained workflow model. Subsequently, the self-contained workflow model can be transferred into the target environment for the workflow execution without the need to transmit additional files. In the target environment, it is passed to the *deployment orchestrator*, which handles the upload of the deployment models to a suitable deployment system (step 2). Thereby, the required deployment system depends on the kind of deployment models that are referenced by the activities. E.g., if they are imperative or declarative and based on a standard, such as TOSCA [15], or a proprietary format. However, the deployment orchestrator can use any deployment system by providing a plugin system to enable easy extensibility.

Before deploying the services, the deployment orchestrator has to select one of the referenced service deployment models per service if multiple alternatives are available. Therefore, the available computing infrastructure can be registered at the deployment orchestrator by a system administrator. This information can be used to exclude deployment models that utilize infrastructure that is not available in the target environment. In case, that no deployment models remain, the deployment must be aborted and the user has to be informed. If multiple alternative deployment models still exist, the selection can be continued by comparing the non-functional requirements, that can be specified by the user, with the non-functional properties of the different deployment models [21].

After selecting the service deployment models, the deployment orchestrator triggers the deployment of all required self-hosted services (step 3). For the creation of the services, different input parameters, such as user name and password for the deployment on a private cloud, can be required. These input parameters have to be provided by the user in step 3. Alternatively, all parameters can already be included in the deployment models that are contained in the self-contained workflow model. This eases the instantiation of the workflow model. However, it can reduce the portability if, e.g., a deployment model using a locally installed hypervisor is part of the self-contained workflow model. Therefore, the endpoint of the hypervisor should be provided by the user after transferring the archive into the target environment. After deploying the services, they are bound to the workflow by the deployment orchestrator. Subsequently, the correctly configured workflow can be deployed into the workflow engine (step 4). Finally, the workflow engine executes the workflow, and the activities access the deployed services (step 5). Thus, the workflow can be executed with no manual task except the upload of the self-contained workflow model to the deployment orchestrator in the target environment despite the usage of self-hosted service.

5 Prototype

This section presents the prototypical implementation of our approach. Due to its wide distribution, the workflow language BPEL and the open-source workflow engine *Apache ODE*[3] were selected to model and execute the workflows. For the specification of the service deployment models, the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [15] is used. TOSCA is an OASIS standard, which allows describing cloud applications in a vendor-neutral way, and therefore, eases portability and interoperability of modeled applications. The prototype is based on the open-source TOSCA modeling tool *Winery*[4]. We extended Winery to enable the modeling of BPEL-based service compositions and the attachment of declarative service deployment models to

[3] https://github.com/apache/ode.
[4] https://github.com/OpenTOSCA/winery.

the activities. The resulting workflow can be packaged by Winery into a self-contained workflow archive. Furthermore, the services and the workflow can be deployed automatically using the workflow engine Apache ODE and the *OpenTOSCA Container*[5], an open-source TOSCA-compliant runtime, which is part of the *OpenTOSCA ecosystem* [1]. After the service deployment is successful, the services can be bound to the workflow by Winery. Therefore, the required services can be deployed and the workflow can be configured fully automatically. The created enhancements are plug-in based and can easily be extended to support other workflow engines or deployment systems.

6 Related Work

Research works from different research areas focus on the development of self-contained archives or packaging formats. The TOSCA [15] standard can be used to define cloud applications in a portable and self-contained manner. For this, all required information for the application deployment is packaged in a *Cloud Service Archive (CSAR)*, which can be executed by any TOSCA-compliant runtime. Qasha et al. [16] provide a framework for scientific workflow reproducibility and portability in the cloud. For this, they propose to use TOSCA to define scientific workflows together with the specification of the hosting environment. Hence, the resulting CSAR can be used by a TOSCA runtime to automatically deploy the required services and to enact the workflow. However, every workflow activity, its execution environment, and control or data connections between different activities have to be modeled using TOSCA. This can lead to a cluttered model that gets incomprehensive. Furthermore, benefits from classical workflow languages and engines, like widely known graphical notations or automatic scaling, can not be reused directly and have to be provided additionally.

Different approaches use *virtual machine images* to provide workflows in a portable and reproducible manner [10,17]. This means, they create virtual machine images from the running services, which can then be utilized to execute the workflow in other environments. For this, new virtual machines are created from each required image. However, the virtual machine images often depend on provider-specific extensions, which are used to improve the performance, and therefore, the portability is reduced. Additionally, services running on other infrastructures, like local workstations, are not considered. Another problem is the size of the virtual machine images that impedes their transmission.

Several scientific workflow management systems provide capabilities to submit tasks to available computing resources and to set up required services automatically when they are invoked by a workflow. *Pegasus* [4] separates the description of the scientific workflow from the execution environment to allow the specification of portable workflows. Additionally, it enables the runtime optimization of workflows regarding the performance or reliability by selecting appropriate

[5] https://github.com/OpenTOSCA/container.

computing resources for the given requirements. Thus, Pegasus contains a *mapper* component that searches and assigns computational resources to activities of the abstract workflow provided by the user. However, Pegasus is only capable to use existing resources and prepares them by transferring files or needed executables. In contrast to our approach, it is not possible to deploy required services using arbitrary deployment models employing Pegasus.

Kepler [19] is an open-source scientific workflow management system, that was extended to enable the usage of EC2 resources within workflows. Therefore, it is possible to deploy services on cloud resources in scientific workflows. However, this extension is provider-specific and not suited for services that should be hosted on different infrastructure. Furthermore, every task, including the deployment of virtual machines, the setup of needed programs and the copying of data, has to be modeled within the workflow. This can quickly lead to a cluttered model that gets incomprehensive and decreases the reusability.

Vukojevic-Haupt et al. [18] introduce an approach for the on-demand deployment of services that are required by a workflow. This means the services are deployed when they are invoked by the workflow and decommissioned afterward to save computing resources. Therefore, they proposed the extension of an enterprise service bus to enable the deployment of services. However, they assume that all required services are available as so-called *service packages* in a local service repository. The service packages include all artifacts needed to deploy the services, and therefore, correspond to deployment models in our approach. Hence, in contrast to our approach, the workflow archives are not self-contained, and new service packages must be registered by the service registry manually before the workflow is initiated, which reduces the portability of the workflows.

7 Conclusion

In this paper, we presented an approach (i) to specify service compositions in a self-contained manner using imperative workflow models and (ii) to support the automatic deployment of services that are required in the environment. For this, we defined self-contained imperative workflow models, which contain the workflow, and additionally, the deployment models of the required services that are attached to the corresponding activities of the workflow. Hence, the user is no longer responsible for determining the required services for a workflow, transferring the corresponding deployment models into the target environment, initiating the deployment, and configuring the workflow with the service endpoints before executing it. Instead, these time-consuming and error-prone tasks can be automated completely. Further, our approach eases the execution of the workflow in another environment, as all required software artifacts are contained in the self-contained imperative workflow model. We prototypically implemented our approach using the TOSCA standard to model declarative deployment models and BPEL as the workflow language to specify the service compositions.

Acknowledgement. This work was partially funded by the DFG project *DiStOPT* (252975529), the DFG's Excellence Initiative project *SimTech* (390740016), and by the BMWi project *Industrial Communication for Factories* – IC4F (01MA17008G).

References

1. Binz, T., et al.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 692–695. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_62
2. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining declarative and imperative cloud application provisioning based on TOSCA. In: International Conference on Cloud Engineering (IC2E), pp. 87–96. IEEE (2014)
3. Bucchiarone, A., Gnesi, S.: A survey on services composition languages and models. In: International Workshop on Web Services-Modeling and Testing, p. 51 (2006)
4. Deelman, E., Vahi, K., Rynge, M., Juve, G., Mayani, R., da Silva, R.F.: Pegasus in the cloud: science automation through workflow technologies. IEEE Internet Comput. **20**(1), 70–76 (2016)
5. Ellis, C.A.: Workflow technology. Comput. Support. Coop. Work Trends Softw. Ser. **7**, 29–54 (1999)
6. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J.: Declarative vs. imperative: two modeling patterns for the automated deployment of applications. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications, pp. 22–27. Xpert Publishing Services (2017)
7. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
8. Freelon, D.G.: ReCal: intercoder reliability calculation as a Web service. Int. J. Internet Sci. **5**(1), 20–33 (2010)
9. Gottschalk, K., Graham, S., Kreger, H., Snell, J.: Introduction to Web services architecture. IBM Syst. J. **41**(2), 170–177 (2002)
10. Jiang, F., Castillo, C., Schmitt, C., Mandal, A., Ruth, P., Baldin, I.: Enabling workflow repeatability with virtualization support. In: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science, p. 8. ACM (2015)
11. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Comput. Surv. (CSUR) **48**(3), 33 (2016)
12. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR, Upper Saddle River (2000)
13. Leymann, F., Roller, D., Schmidt, M.T.: Web services and business process management. IBM Syst. J. **41**(2), 198–211 (2002)
14. OASIS: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Organization for the Advancement of Structured Information Standards (OASIS) (2007)
15. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
16. Qasha, R., Cała, J., Watson, P.: A framework for scientific workflow reproducibility in the cloud. In: IEEE 12th International Conference on e-Science (e-Science), pp. 81–90. IEEE (2016)
17. Stodden, V., Leisch, F., Peng, R.D.: Implementing Reproducible Research. CRC Press, Boca Raton (2014)

18. Vukojevic-Haupt, K., Karastoyanova, D., Leymann, F.: On-demand provisioning of infrastructure, middleware and services for simulation workflows. In: Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications, pp. 91–98. IEEE (2013)
19. Wang, J., Altintas, I.: Early cloud experiences with the Kepler scientific workflow system. Procedia Comput. Sci. **9**, 1630–1634 (2012)
20. Wurster, M., et al.: The essential deployment metamodel: a systematic review of deployment automation technologies. Softw. Intensive Cyber Phys. Syst., 1–13 (2019). https://doi.org/10.1007/s00450-019-00412-x
21. Yu, T., Lin, K.J.: Service selection algorithms for Web services with end-to-end QoS constraints. IseB **3**(2), 103–126 (2005)