Combining TOSCA and BPMN to Enable Automated Cloud Service Provisioning

Domenico Calcaterra, Vincenzo Cartelli, Giuseppe Di Modica and Orazio Tomarchio
Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy

Keywords: Cloud Provisioning, Cloud Orchestration, TOSCA, BPMN.

Abstract: The Cloud computing paradigm has kept its promise to transform computing resources into utilities ready to be consumed in a dynamic and flexible way, on an "as per need" basis. The next big challenge cloud providers are facing is the capability of automating the internal operational processes that need to be run in order to efficiently serve the increasing customers' demand. When a new cloud service request has to be served, there is a bunch of operations the provider needs to carry out in order to get the requested cloud service up and ready for usage. This paper investigates the automation of the "provisioning" activities that must be put into place in order to build up a cloud service. Those activities range from the procurement of computing resources to the deployment of a web application, passing through the installation and configuration of third party softwares and libraries that the web application depends upon in order to properly work. Leveraging on a well-known specification used for the representation of a cloud application's structure (TOSCA), we designed and implemented an orchestrator capable of automating and putting in force, in the correct timing, the sequence of tasks building up the cloud application in a step-by-step fashion. The novelty in the followed approach is represented by the definition of a converter which takes as input a TOSCA template and produces a workflow that is ready to be executed by a workflow engine. The BPMN notation was used to represent both the workflow and the data that enrich the workflow. To support the viability of the proposed idea, a use case was developed and discussed in the paper.

1 INTRODUCTION

In the past five years the scientific community has shown a growing interest around the topic of cloud provisioning and orchestration (Ranjan et al., 2015; Bousselmi et al., 2014). The appeal of this topic is further witnessed by the investments that big cloud players have been making to develop tools and softwares that support the automation of cloud services' delivery and maintenance. Also, many commercial players have been engaged in the definition of international standards that would foster the widespread adoption of technological solutions for the orchestration of portable (i.e., provider-agnostic) cloud applications.

In the panorama of standard initiatives, OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) (OASIS, 2013) has become very popular. It is supported by many big cloud players and promises to cater for the cloud providers' need of streamlining cloud service orchestration and provisioning operations. Also, the standardization body has released a version of the specification which makes use of a very simple and human understandable language (YAML) that has contributed to speed-up the standard adoption process.

The work described in this paper grounds on the TOSCA specification. It leverages the TOSCA features to build up a cloud service orchestrator capable of automating the execution of tasks and operations required for the provisioning of a cloud application. The strategy adopted by the cloud orchestrator is to convert a TOSCA cloud application model into its equivalent BPMN workflow and dataflow model (OMG, 2011). The orchestrator will then use a BPMN engine to enforce the operations specified in the BPMN model. The approach we propose clearly separates the orchestration of the provisioning tasks from the real provisioning services (i.e., the e-services that enforce the provisioning). In this paper, we present the design of a cloud service provisioning framework, and discuss the design and implementation of a cloud orchestrator prototype. Further, we discuss a real use case of a cloud application provisioning.

The remainder of the paper is organized in the fol-

Calcaterra, D., Cartelli, V., Modica, G. and Tomarchio, O.
Combining TOSCA and BPMN to Enable Automated Cloud Service Provisioning.
DOI: 10.5220/0006304701870196
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 159-168
ISBN: 978-989-758-243-1

lowing way. In Section 2 a survey of the literature is proposed. Section 3 provides a bird's eye view of the TOSCA specification. The core ideas of this work, along with the design and implementation details of the cloud orchestrator, are discussed in Section 4. A real use case showing the potential of the proposed idea is discussed in Section 5. Section 6 draws some final considerations and suggests some future directions.

2 RELATED WORK

This section presents a survey of all the recent and authoritative initiatives, both commercial and scientific, that address the cloud provisioning and orchestration topic.

Many **cloud industry** players have developed cloud management platforms (Cisco, 2016; Amazon, 2016; Rightscale, 2016; RedHat, 2016; HP, 2016; IBM, 2016; GigaSpaces, 2016) for automating the provisioning of cloud services. All platforms, to varying degrees, promise to provide automation in three fundamental steps: *cloud configuration*, *cloud provisioning* and *cloud deployment*. The more advanced platforms also offer services and tools for the management of cloud applications' lifecycle. None of these commercial products are open to the community, and the solutions they offer are not portable across third-party providers either.

The open source world has shown interest on this topic as well. Taking a look at the category of **configuration management** tools, DevOps Chef (Chef, 2016) is a software used to streamline the task of configuring and maintaining server applications and utilities. It is based on the concept of configuration "recipes", which are instructions on the desired state of resources (software packages to be installed, services to be run, or files to be written). Chef takes care of those recipes and makes sure that resources are actually in the desired state. Chef can integrate with cloud-based platforms such as Amazon EC2, Google Cloud Platform, OpenStack, Microsoft Azure and Rackspace to automatically provision and configure new virtual machines. Similar recipe-based approaches are proposed by other open-source solutions like Puppet (Puppet, 2016) with its *Puppet manifests* and Juju (Juju, 2016) with its *Juju charms*. Speaking of **cloud orchestration** tools, OpenStack Heat (OpenStack, 2016) is a service to orchestrate composite cloud applications using a declarative template format - namely, the Heat Orchestration Template (HOT) - through both an OpenStack-native REST API and AWS CloudFormation-compatible API (Amazon, 2016). HOT describes the infrastructure for a cloud application in text files which are readable and writable by humans and by software tools as well. Also, it integrates well with software configuration management tools such as Puppet and Chef. Very recently, orchestration concepts have been analyzed also in the context of *containers* (Tosatto et al., 2015). Even if containers represent a portable unit of deployment, when an application is built out of multiple containers the setting up of a cluster of containers can become actually complex, because it is needed to make one container aware of another and expose several details required for them to communicate. As an example, Docker Compose, currently under active development, is one of the first tools for defining and running multi-container Docker applications (Docker, 2017).

With respect to **standardizing initiatives**, OASIS is the most active on the topic. TOSCA (OASIS, 2013) is an OASIS open cloud standard supported by a large and growing number of international industry leaders. It defines an *interoperable description* of applications, including their components, relationships, dependencies, requirements, and capabilities, thus enabling *portability* and *automated management* across multiple cloud providers regardless of underlying platform or infrastructure. No commercial solution supports processing of the TOSCA specification at this moment. OpenTOSCA (Binz et al., 2013) is a famous open source TOSCA runtime environment. Although authors have been working on adding support to the TOSCA Simple Profile (OpenTOSCA, 2015), only a few YAML elements are supported by the converter. At this moment, imports, inputs, outputs and groups are not supported, thereby limiting the description of application components. The reader may find some insight on the technical aspects of TOSCA in Section 3.

In the **scientific literature** a few works have addressed the TOSCA specification. In (Kopp et al., 2012), *BPMN4TOSCA* was proposed as a domain-specific BPMN (OMG, 2011) extension to ease modeling of management plans by enabling convenient integration and direct access to TOSCA topology and provided management operations. Since the BPMN4TOSCA extension introduces new functionalities which are not natively supported by workflow engines, it leads to a non-standards-compliant BPMN and, therefore, needs special treatment, i.e., a transformation to plain BPMN. In (Katsaros et al., 2014), a *proof of concept* for the actual portability features of TOSCA on OpenStack and Opscode Chef has been presented. To that end, an execution runtime environment named *TOSCA2Chef* was developed to automate the deployment of TOSCA-based cloud application topologies to OpenStack by employing Chef and BPEL processes. In (Wettinger et al., 2014), a unified invocation bus and interface to be used by TOSCA management plans has been presented. Based on OpenTOSCA's architecture, a service bus (*Operation Invoker*) was implemented to provide a unified invocation interface for TOSCA plans to invoke operations. In (Wettinger et al., 2016), with the goal of achieving a seamless and interoperable orchestration of arbitrary artifacts, an integrated modelling and runtime framework has been introduced. After executable DevOps artifacts of different kinds get discovered and stored in DevOps knowledge repositories, they are transformed into TOSCA-based descriptions. In (Breitenbücher et al., 2015), a process modelling concept to enable the integration of provisioning models has been introduced. The general modelling approach is based on extending imperative workflow languages such as BPMN and BPEL (OASIS, 2007) by means of *Declarative Provisioning Activities*, which enable to specify declarative provisioning goals directly in the control flow of a workflow model. The data flow between provision activities is defined through *input parameters*, *output parameters* and *content injection*. A prototype based on the OpenTOSCA ecosystem and the BPEL workflow language was implemented.

Several EU funded research projects, such as ARTIST (Menychtas et al., 2014), SeaClouds (Brogi et al., 2015), PaaSage (Rossini, 2016), MODAClouds (Ferry et al., 2017) and PaaSport (Bassiliades et al., 2017), also addressed cloud application portability in its essence. Most of these projects, instead of building a TOSCA engine, transform the TOSCA-based application specification into a single orchestration script, such as YAML, and execute it by a corresponding management tool, such as CAMP (OASIS, 2014), Brooklyn (The Apache Software Foundation, 2016), etc.

The work we propose grounds on the TOSCA standard as well. A distinctive feature of our approach is the clear separation between the *orchestration of the provisioning tasks*, intended as the scheduling of the logical steps to be taken, and the *provisioning services*, which are the services implementing the tasks' instructions. As for the orchestration aspect, we devised a mechanism that automatically builds a plain BPMN orchestration plan starting from a cloud application's TOSCA model.

3 THE TOSCA SPECIFICATION

TOSCA is the acronym for *Topology and Orchestration Specification for Cloud Applications*. It is a standard put together by OASIS that can be used to enable the portability of cloud applications and related IT services. This specification permits describing the structure of a cloud application as a *service template*, that is in turn composed of a *topology_template* and the types needed to build such a template. The topology_template is a typed directed graph, whose nodes (called *node_templates*) model the application components, and edges (called *relationship_templates*) model the relations occurring among such components. Each node of a topology can also be associated with the corresponding component's requirements, the operations to manage it, the capabilities it features, and the policies applied to it. Inter-node dependencies associate the requirements of a node with the capabilities featured by other nodes. TOSCA supports the deployment and management of applications in two different flavors: *imperative processing* and *declarative processing*. The imperative processing requires that all needed management logic is contained in the *Cloud Service Archive (CSAR)*, which stores all software artifacts required to provision, operate, and manage the application. *Management plans* imperatively orchestrate low-level management operations that are either provided by the application components themselves or by publicly accessible services (e.g., the Amazon Web Services API). These plans are typically implemented using workflow languages, such as BPMN or BPEL (OASIS, 2007). The declarative processing shifts management logic from plans to runtime, therefore no plans are actually required. TOSCA runtime engines automatically infer the corresponding logic by interpreting the application topology template. This requires a precise definition of the semantics of nodes and relations based on well-defined *Node Types* and *Relationship Types*. The set of provided management functionalities depends on the corresponding runtime and is not standardized by the TOSCA specification.

The TOSCA Simple Profile is a rendering of the TOSCA specification in the YAML language (OASIS, 2015). It aims to provide a more accessible syntax as well as a more concise and incremental expressiveness of the TOSCA language in order to speed up the adoption of TOSCA to describe cloud applications in a portable manner. The work described in this paper heavily grounds on the TOSCA standard and, specifically, on the TOSCA Simple Profile.

4 DESIGN OF A CLOUD SERVICE PROVISIONING FRAMEWORK

This work addresses the design and implementation of a software framework that aims at easing and automating the processes that support the operational management of cloud services. The stakeholders that may have interest in the services provided through the framework are the Customers in need of cloud resources and cloud applications (in a nutshell, "cloud services") and the Providers of cloud services. To the former, the framework offers tools to clearly state functional requirements of the cloud service they are in need of; from those requirements, the framework puts in force the actions necessary for the service delivery to take place. The latter have the chance to offer their cloud services through the framework, while playing no active role in the service orchestration which, instead, is in charge of the framework itself.

The focus of this work is put on the automation of the **cloud service provisioning process**, i.e., the process which is entrusted with the procurement and the set-up of all the resources that build up the cloud service requested by the Customer. We point out that the framework has been designed to integrate the support for more sophisticated operations such as, to cite a few, resource monitoring, resilience and scaling. Those specific operations are though out of the scope of the current work, and will be part of our future work's investigation. As for the service provision, the objective of this work will be the design and implementation of an orchestrator which, starting from the Customer requirements, is capable of generating on the fly a *cloud provisioning process* made up of tasks that build up the ready-to-use service to be delivered. Specifically, the orchestrator will be in charge of coordinating the overall process by making sure that every task's activity is carried out in accordance with the proper timing.

We surveyed the literature in search of a well established and broadly accepted way of representing the cloud application requirements, i.e., a language or a meta-model the Customer may use to express the stack of resources (from the virtual machine up to the top level libraries and softwares) they need in order to set up their application, and also, the way those resources need to get configured and coupled together in order to ensure that the final service delivered to the Customer will meet the Customer expectation. As mentioned before, the approach we propose grounds on the *OASIS TOSCA* standard and, more specifically, on the *TOSCA Simple Profile* rendering. The choice of TOSCA was driven by the fact that TOSCA is a mature standard which embeds all the features which we deem useful to our purpose. In particular, the TOSCA Simple Profile provides a meta-model written in YAML (a human friendly data serialization standard) which the Customer may use to define their **cloud application model**, i.e., to describe both the application topology and the artifacts needed by the application itself. Since the objective we pursue is to automate the application provisioning, we opted for a *workflow-based* solution which, starting from the cloud application model description, is capable of devising and orchestrating the flow of the provisioning operations to execute. Instead of developing a workflow engine from scratch, we decided to make use of a BPMN engine, i.e., an engine capable of executing workflows represented in the BPMN language. Since a YAML application model is not executable by a BPMN engine, we developed an ad-hoc YAML-to-BPMN converter. The reader may discover the details of the converter in Section 4.1. We opted on the BPMN as workflow language since it is a robust standard and it also provides support for data modelling, a feature that we exploited to represent the application artifacts needed along the workflow.

A novelty introduced by this approach is the separation between the orchestration of the provisioning tasks and the provisioning services themselves. We propose a solution where the provisioning services may be supplied by third party service providers, while the provisioning tasks orchestrated by the workflow engine will draw on those services in a SOA (Service Oriented Architecture) fashion. This enables a scenario of a market of services in which many providers are allowed to participate and where Customers can get the best combination of services that meet their requirements. The overall scenario described so far is best depicted in Figure 1.

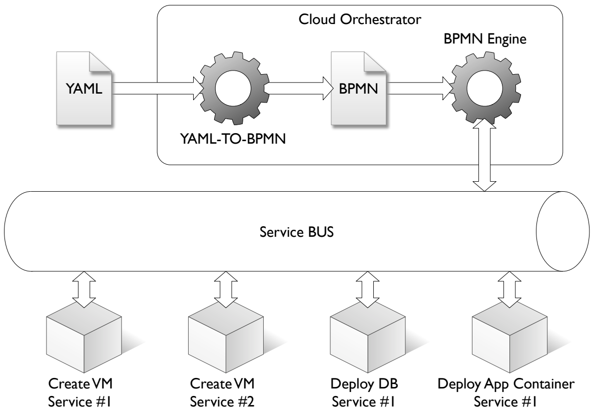

Figure 1: Cloud Orchestrator scenario.

We have designed and implemented a **TOSCA**

Orchestrator which takes as input the application model and deploys the concrete application in the cloud. The Orchestrator takes the YAML model and transforms it into an equivalent BPMN model. The BPMN model, in its turn, is fed to a BPMN engine that will instantiate and coordinate the relative process. The process will put in force all the provisioning activities needed to build up the application stack (e.g., getting a virtual machine from a cloud provider, installing all the required libraries and third party softwares on it, configuring the software dependencies, and so on); as the reader may notice, the provisioning activities access a service bus in order to get the required services which, in their turn, are supplied by third party service providers. Finally, once the cloud application is up and running, the Customer is invited to take the control.

The framework we propose aims to offer tools and services that enable the scenario depicted in Figure 1. To date, only a subset of those services has been implemented. Specifically, the implementation of the service bus and the invocation of the provisioning services will be addressed in future work. Customers can use the YAML representation to express application requirements and push those requirements to the framework. Providers can design their services according to specific templates and offer them to Customers through the framework. The framework is entrusted with orchestrating the provisioning activities and matching the services' offer and demand. In the current implementation, the framework cares just for functional requirements, i.e., it provides matches between what the Customer needs in terms of functional needs (gets a given virtual machine, installs a specific database, etc.). Non-functional requirements, which call for enhanced service matchmaking mechanisms, are out of the scope of this work and will be addressed in future work.

4.1 Converting YAML to BPMN

This section discusses the features and the technical details of the software component we devised to convert a TOSCA Simple Profile into its equivalent BPMN process model. Starting from a TOSCA Simple Profile compliant service template, our software creates a *Provisioning Plan* which is fed into the workflow engine for the automated application deployment. This approach brings considerable benefits, among which a) reusability of the process logic, since components of the same type use the same logic; b) portability of the Plan, as the application can be deployed on a generic Cloud Provider; c) efficiency in terms of streamlining Customer's work, because they only have to define their templates and fill them with the management functions of their choice, without caring about how Provisioning Plans will be created and executed on the Cloud Provider.

The proposed solution consists of three components: TOSCA-Parser, BPMN-Generator, and BPMN-Validator. The *TOSCA-Parser* deals with the service template by providing means to load, parse and validate the YAML file, and creates the *dependency graph*, a data structure containing the relationships between all of the nodes in the TOSCA template. Vertices in the graph represent Nodes, while edges represent relationships occurring between them. The *BPMN-Generator* grounds the creation of the Provisioning Plan on the parsed service template and the dependency graph. The *BPMN-Validator* validates the automatically generated Plan against the BPMN specification. The following Sections will provide more details about these components.

4.1.1 TOSCA-Parser

The TOSCA Parser takes a TOSCA YAML template as input, with an optional dictionary of needed parameters with their values, validates it, and produces in-memory objects of different TOSCA elements with their relationship to each other. It also creates an in-memory graph of TOSCA node templates and their relationships. This software component is widely based on the OpenStack parser for TOSCA Simple Profile in YAML (OpenStack, 2016), a Python project licensed under Apache 2.0. In agreement with the overall structure of a *service template*, the parser contains various Python modules to handle it including topology templates, node templates, relationship templates, data types, node types, relationship types, capability types, artifact types, etc. The *ToscaTemplate* class is an entry class of the parser and is of great importance, along with *TopologyTemplate*, *NodeTemplate* and *RelationshipTemplate*, in the construction of the *ToscaGraph*, which keeps track of all nodes and dependency relationships between them in the TOSCA template. This in-memory graph is, in its turn, a milestone in the generation of the BPMN Provisioning Plan, and the entire process is covered in Section 4.1.2.

4.1.2 BPMN-Generator

The BPMN-Generator takes the aforementioned *ToscaGraph* and *ToscaTemplate* elements (e.g., Inputs, Outputs, NodeTemplates, RelationshipTemplates) as input and automatically generates the BPMN Provisioning Plan for the designated Cloud application. For clarity purposes, the service template shown in Listing 1 will be taken as an example to show what needs to be done to reach the goal. The BPMN generation is composed of the following two steps: (1) the creation of a *Workflow* modelling a detailed sequence of business activities to perform; (2) the creation of a *Dataflow* modelling the data to be read, written or updated during the Workflow execution.

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: >
  TOSCA simple profile with a software component.

topology_template:
  inputs:
    cpus:
      type: integer
      description: Number of CPUs for the server.
      constraints:
        - valid_values: [ 1, 2, 4, 8 ]
      default: 1

  node_templates:
    sw:
      type: tosca.nodes.SoftwareComponent
      properties:
        component_version: 1.0
      requirements:
        - host: server
      interfaces:
        Standard:
          create: software_install.sh
          start: software_start.sh

    server:
      type: tosca.nodes.Compute
      capabilities:
        host:
          properties:
            disk_size: 10 GB
            num_cpus: { get_input: cpus }
            mem_size: 1024 MB
        os:
          properties:
            architecture: x86_64
            type: Linux
            distribution: Ubuntu
            version: 14.04
```

Listing 1: SW Component - Service Template.

The Workflow basically comprises a BPMN process made of Service Tasks, Sequence Flows and Gateways used to control how the process flows, with every single component being derived from all the node_templates and their requirements in the YAML Service Template. In particular, taking inspiration from normative node states and lifecycle operations of the Standard interface (OASIS, 2013), each node_template in the YAML scenario leads to a new Service Task for every operation specified on that node. Such Service Tasks are related to each other by means of Sequence Flows and possible Gateways, whose creation depends on Service Tasks dependencies, which, in their turn, depend on node_templates requirements. The *ToscaGraph* is the reference point to determine such requirements. In this regard, the graph is traversed and for each node, represented by a vertex, the whole set of requirements is constructed in terms of relationships with other nodes, represented by related edges. Service Tasks dependencies are then obtained by taking into account the node requirements and the lifecycle operations they represent. Starting from such dependencies, it is possible to compute the execution order of all Service Tasks in the Provisioning Plan, i.e., the deployment order of all Cloud application components. This information is represented by numerical data: the lower the number is, the less priority that Service Task gets. Service Tasks with the lowest execution order, hereby collectively called *Service Tasks Endpoint*, don't feature in between any Service Task's required dependencies, whereas Service Tasks with the highest execution order, hereby collectively called *Service Tasks Startpoint*, don't feature any Service Task as a required dependency. With reference to our example scenario, the resulting data structures are shown in Listing 2.

```
service_tasks = ['server', 'sw_create', 'sw_configure',
        'sw_start']

service_tasks_requirements = {'sw_create': ['server'],
        'sw_configure': ['sw_create'],
        'sw_start': ['sw_configure']}

service_tasks_order = {'server': 4, 'sw_create': 3,
        'sw_configure': 2, 'sw_start': 1}

service_tasks_startpoint = ['server']

service_tasks_endpoint = ['sw_start']
```

Listing 2: SW Component - Tasks, Requirements, Order.

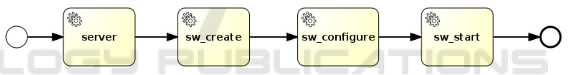

Figure 2: Output of the BPMN generator - Workflow.

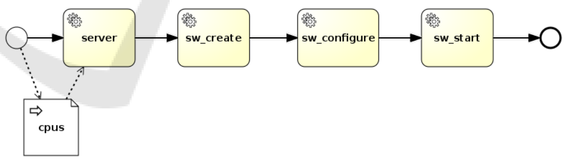

Figure 3: Output of the BPMN generator - Workflow and Dataflow.

Service Tasks Endpoint and Startpoint are of paramount importance to define a proper execution flow, because they may lead to some degree of parallelism in the Workflow through *Parallel Gateways*, which are used to synchronize or create parallel flows. Specifically, they play a role in the creation of *Start Event*, *End Event* and *Service Tasks*. Service Tasks and related Sequence Flows are created by proceeding in ascending Service Tasks priority fashion (i.e., in their reverse execution order). From lowest to highest priority, each Service Task is created and then their incoming and outgoing paths are determined by distinguishing three further cases: a) the Service Task

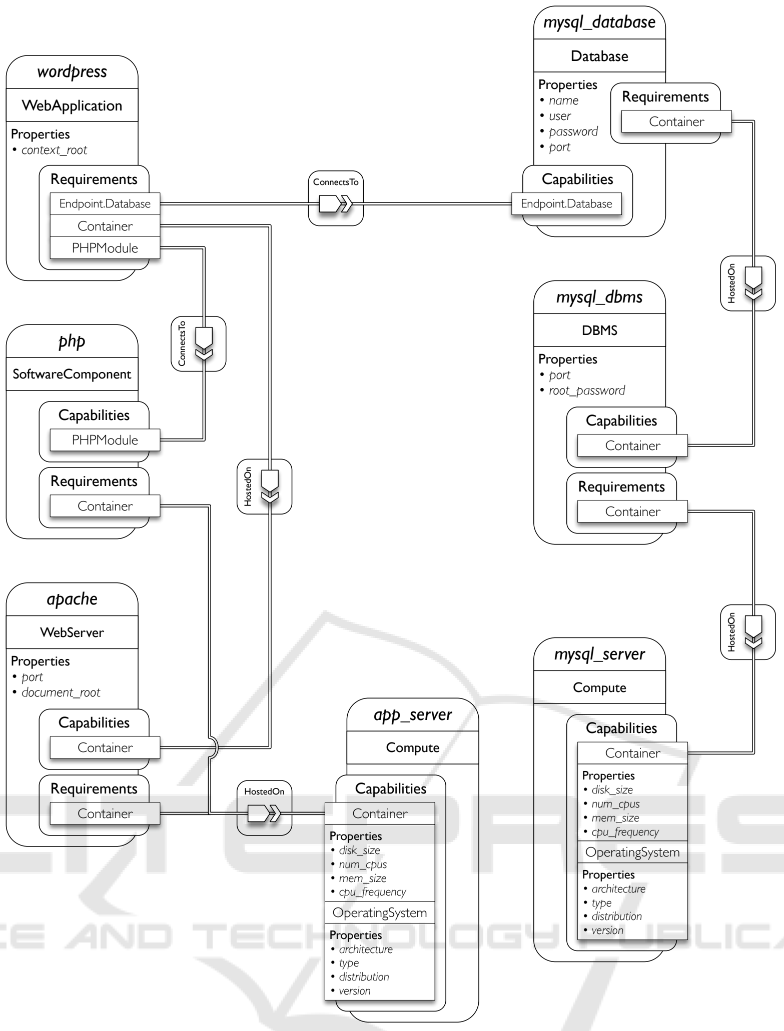

Figure 4: Wordpress Deploy - TOSCA Template.

belongs to Service Tasks Endpoint set, b) the Service Task belongs to Service Tasks Startpoint set, c) the Service Task belongs to neither of them. As to our example scenario, the resulting BPMN Workflow is shown in Figure 2.

The Dataflow simply consists of Data Inputs, Data Outputs and Data Objects, which are derived from node_templates and their data requirements in the YAML Service Template. Speaking of data requirements, the TOSCA standard allows template authors to customize Service Templates through the *inputs* section in the Topology Template, which represents an optional list of input parameters for the Topology Template. In a complementary way, the *outputs* section represents an optional list of output parameters for the Topology Template. Inputs and outputs can be used to parameterize node_templates properties or node_templates and relationship_templates lifecycle operations. Data Inputs, which capture input data that Activities and Processes often need in order to execute, are utilized to model such inputs; Data Outputs, which capture data that they can produce during or as a result of execution, are utilized to model such outputs. It should be noted that node_templates *attributes* can be used as parameters in the lifecycle operations as well. Data Objects are utilized to model this kind of data requirements, with Data Associations determining how information stored in Data Objects is handled and passed between Process flow elements. With reference to our sample template in Listing 1, there is only one input variable specified in the server *num_cpus* property. This leads to a Data Input and a Data Association between the Start Event and the server Service Task, as depicted in Figure 3.

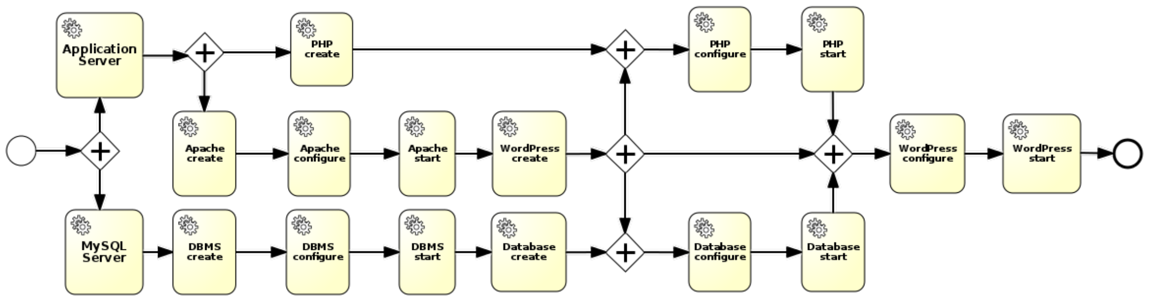

Figure 5: Wordpress Deploy - Workflow.

4.1.3 BPMN-Validator

The BPMN-Validator validates the BPMN Plan generated in the previous step against the BPMN XML Schema (OMG, 2011), with both of them being taken as input parameters. The validation is performed by means of *etree* module in Python *lxml* package (lxml, 2016). More specifically, the BPMN XML Schema gets parsed and turned into an XML Schema validator, which checks if the previously parsed BPMN plan complies with the provided schema. If that is not the case, then a validation error is going to be raised.

5 USE CASE

The Application modelling use case taken into consideration aims to deploy a WordPress web application on an Apache web server, with a MySQL DBMS hosting the database content of the application on a separate server. Figure 4 shows the overall architecture compliant with the TOSCA Simple Profile specification (although wordpress, php and apache node types are non-normative). There are two separate servers: *app_server* for the web server hosting and *mysql_server* for the DBMS hosting. Both servers are configurable on *hardware side* (e.g., disk size, number of cpus, memory size and CPU frequency) and *software side* (e.g., OS architecture, OS type, OS distribution and OS version). The *apache* node features *port* and *document_root* properties, and is dependent upon the app_server via a *HostedOn* relationship as well. In the same way, the *php* node is dependent upon the app_server via a *HostedOn* relationship. The *mysql_dbms* node features *port* and *root_password* properties, and a *HostedOn* dependency relationship upon the mysql_server. The *mysql_database* node features *name*, *username*, *password* and *port* properties, and a *HostedOn* dependency relationship upon the mysql_dbms. Finally, the *wordpress* node features the *context_root* property, and depends on mysql_database and php by means of two *ConnectsTo* relationships and on apache by means of a *HostedOn* relationship, respectively.

```
apache:
  type: tosca.nodes.WebServer.Apache
  properties:
    port: { get_input: apache_port }
    document_root: { get_input: apache_doc_root }
  requirements:
    - host: app_server
  interfaces:
    Standard:
      create:
        inputs:
          ip: { get_attribute: [app_server, private_address]}
          port: { get_property: [SELF, port] }
          doc_root: { get_property: [SELF, document_root]}
        implementation: scripts/install_apache.sh
      start:
        inputs:
          ip: { get_attribute: [app_server, private_address]}
        implementation: scripts/start_apache.sh
```

Listing 3: Wordpress Deploy - Apache node.

For the sake of clarity, Listing 3 shows the apache node declaration in YAML. As mentioned above, the node takes the app_server as requirement and has *port* and *document_root* properties, whose values are retrieved from *apache_port* and *apache_doc_root* input parameters, respectively, by means of the *get_input* intrinsic function. Two lifecycle operations are also defined (i.e., *create* and *start*), with both of them taking *ip* as input parameter, whose value is retrieved from the *private_address* attribute of the app_server through the *get_attribute* intrinsic function. In conformity with Section 4.1.2, the node transformation from YAML to BPMN leads to the creation of: (1) three Service Tasks (*apache_create*, *apache_configure* and *apache_start*); (2) two Data Inputs (*doc_root* and *port*) with their respective Data Input Associations in apache_create; (3) one Data Object (*app_server.private_address*) with its Data Input Associations in apache_create and apache_start. Figure 5 and Figure 6 show the overall Workflow and Dataflow-decorated Workflow, respectively. The workflow represented in Figure 6 is then fed to a BPMN engine that will actually enforce the workflow's tasks.

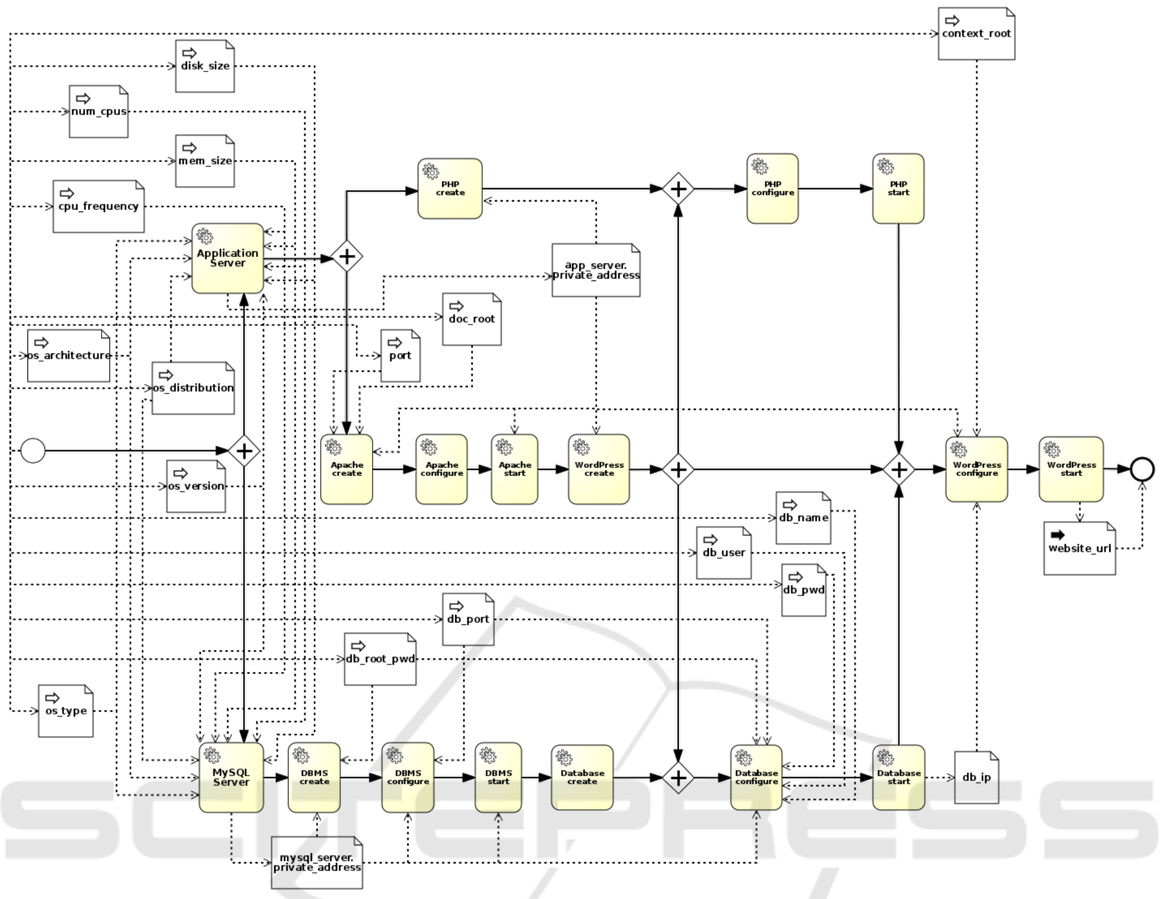

Figure 6: Wordpress Deploy - Workflow and data flow.

6 CONCLUSION

The challenge of automating the operational management and orchestration of cloud services has attracted many cloud industry players. The open source world too is very active in this topic. Very recently an open standard for the representation of the cloud application structure (TOSCA) has boosted the interest in the research environment and has fostered the flourishing of many more cloud automation products even from small cloud players. This work leverages on the TOSCA potential to propose the definition of a cloud orchestration and provisioning framework that automates the cloud service deployment operations. Basically, the automation is carried out by a two-step process: 1) transforming a TOSCA cloud application model into a BPMN workflow; 2) getting the workflow executed on a workflow engine. The novelty of the approach also consists in the definition of a data model that enriches the workflow. In the future, the framework will be enhanced with an ecosystem of services that can be invoked by the workflow engine and that will actually carry out the deployment tasks.

REFERENCES

Amazon (2016). Amazon CloudFormation. https://aws.amazon.com/cloudformation/. Last accessed on 15-02-2017.

Bassiliades, N., Symeonidis, M., Meditskos, G., Kontopoulos, E., Gouvas, P., and Vlahavas, I. (2017). A semantic recommendation algorithm for the paasport platform-as-a-service marketplace. *Expert Systems with Applications*, 67:203 – 227.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). *OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications*, pages 692–695. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bousselmi, K., Brahmi, Z., and Gammoudi, M. M. (2014). Cloud services orchestration: A comparative study of existing approaches. In *IEEE 28th International Con-*

ference on Advanced Information Networking and Applications Workshops, (WAINA 2014), pages 410–416.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2015). A modelling concept to integrate declarative and imperative cloud application provisioning technologies. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, pages 487–496.

Brogi, A., Carrasco, J., Cubo, J., Nitto, E. D., Durán, F., Fazzolari, M., Ibrahim, A., Pimentel, E., Soldani, J., Wang, P., and D'Andria, F. (2015). Adaptive management of applications across multiple clouds: The seaclouds approach. *CLEI Electron. J.*, 18(1).

Chef (2016). Devops Chef. https://www.chef.io/solutions/devops/. Last accessed on 15-02-2017.

Cisco (2016). Cisco Intelligent Automation for Cloud (IAC). http://www.cisco.com/c/en/us/products/cloud-systems-management/intelligent-automation-cloud/index.html. Last accessed on 15-02-2017.

Docker (2017). Docker Compose. https://docs.docker.com/compose/. Last accessed on 15-02-2017.

Ferry, N., Almeida, M., and Solberg, A. (2017). *The MODAClouds Model-Driven Development*, pages 23–33. Springer International Publishing, Cham.

GigaSpaces (2016). Cloudify. http://getcloudify.org/. Last accessed on 15-02-2017.

HP (2016). HP Cloud Service Automation. http://www8.hp.com/it/it/software-solutions/cloud-service-automation/. Last accessed on 15-02-2017.

IBM (2016). IBM Cloud Orchestrator. http://www-03.ibm.com/software/products/it/ibm-cloud-orchestrator. Last accessed on 15-02-2017.

Juju (2016). Juju charms. https://jujucharms.com/. Last accessed on 15-02-2017.

Katsaros, G., Menzel, M., Lenk, A., Revelant, J. R., Skipp, R., and Eberhardt, J. (2014). Cloud application portability with tosca, chef and openstack. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 295–302, Washington, DC, USA. IEEE Computer Society.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2012). *BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications*, pages 38–52. Springer Berlin Heidelberg, Berlin, Heidelberg.

lxml (2016). lxml project. http://lxml.de/. Last accessed on 15-02-2017.

Menychtas, A., Konstanteli, K., Alonso, J., Orue-Echevarria, L., Gorroñogoitia, J., Kousiouris, G., Santzaridou, C., Brunelière, H., Pellens, B., Stuer, P., Strauß, O., Senkova, T., and Varvarigou, T. A. (2014). Software modernization and cloudification using the artist migration methodology and framework. *Scalable Computing: Practice and Experience*, 15(2).

OASIS (2007). Web Services Business Process Execution Language Version 2.0. https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm. Last accessed on 15-02-2017.

OASIS (2013). Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Last accessed on 15-02-2017.

OASIS (2014). Cloud Application Management for Platforms Version 1.1. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html. Last accessed on 15-02-2017.

OASIS (2015). TOSCA Simple Profile in YAML Version 1.0. http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd01/TOSCA-Simple-Profile-YAML-v1.0-csprd01.html. Last accessed on 15-02-2017.

OMG (2011). Business Process Model and Notation (BPMN 2.0). http://www.omg.org/spec/BPMN/2.0/. Last accessed on 15-02-2017.

OpenStack (2016). OpenStack Heat. https://wiki.openstack.org/wiki/Heat. Last accessed on 15-02-2017.

OpenStack (2016). OpenStack project. https://github.com/openstack/tosca-parser. Last accessed on 15-02-2017.

OpenTOSCA (2015). OpenTOSCA project. https://github.com/CloudCycle2/YAML_Transformer. Last accessed on 15-02-2017.

Puppet (2016). Puppet. https://puppet.com/. Last accessed on 15-02-2017.

Ranjan, R., Benatallah, B., Dustdar, S., and Papazoglou, M. P. (2015). Cloud Resource Orchestration Programming: Overview, Issues, and Directions. *IEEE Internet Computing*, 19:46–56.

RedHat (2016). RedHat CloudForms. https://www.redhat.com/it/technologies/management/cloudforms. Last accessed on 15-02-2017.

Rightscale (2016). Rightscale Cloud Management Platform. http://www.rightscale.com/why-cloud-management-platform/benefits. Last accessed on 15-02-2017.

Rossini, A. (2016). *Cloud Application Modelling and Execution Language (CAMEL) and the PaaSage Workflow*, pages 437–439. Springer International Publishing, Cham.

The Apache Software Foundation (2016). The Apache Brooklyn project. https://brooklyn.apache.org/. Last accessed on 15-02-2017.

Tosatto, A., Ruiu, P., and Attanasio, A. (2015). Container-Based Orchestration in Cloud: State of the Art and Challenges. In *9th International Conference on Complex, Intelligent, and Software Intensive Systems, (CISIS 2015)*, pages 70–75.

Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., and Zimmermann, M. (2014). Unified invocation of scripts and services for provisioning, deployment, and management of cloud applications based on tosca. In *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, pages 559–568.

Wettinger, J., Breitenbücher, U., Kopp, O., and Leymann, F. (2016). Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel . *Future Generation Computer Systems*, 56:317 – 332.