P-TOSCA Portability of SOA Applications

Marjan Gusev, Magdalena Kostoska, Sasko Ristov and Aleksandar Donevski
University Ss. Cyril and Methodius, Faculty of Computer Sciences and Engineering, 1000 Skopje, Macedonia

Keywords: Application Portability, SOA, Cloud Computing.

Abstract: Even more frequently, the customers express their increasing need to change the cloud provider and/or the operating cloud environment in order to avoid vendor lock-in. We analyze portability as the transferability of an application from on-premise onto a cloud (migration) and among different clouds (porting). The contribution of this paper is twofold: 1) demonstration of the P-TOSCA model for automated migration and porting of SOA applications onto a cloud and/or switch between cloud providers, and 2) evaluation of a significant time reduction in migration and porting.

1 INTRODUCTION

The Service Oriented Architecture (SOA) is commonly used architecture in the past few years, especially for enterprise applications. It became popular due to the benefits and flexibilities of integrating loosely-coupled and reusable services (Erl, 2004)

Cloud offers a scalable and elastic environment for hosting SOA applications as computing utilities (Buyya et al., 2009). It saves not only companies' OPEX and CAPEX, but also management and administration costs (Rana, 2014). As more customers adopt and use cloud technologies, they encounter turbulence along the increasing number of SOA applications hosted on clouds. A variety of cloud service providers (CSPs) with different service level agreements (SLAs) are now available on the market.

Customers appreciate being able to switch among CSPs and environments, so as to choose the most suitable one to their needs. The ability of a software to run on different cloud platforms in general defines the cloud portability. It presents a hot topic research challenge, although CSPs do not think to offer mechanisms to enable migration of applications onto clouds or to enable a possibility to transfer them between clouds. Mostly, CSPs are stuck to the philosophy that the best solution on the market will become a standard. However, we analyze a possibility to define a specification and build an engine that will enable portability and migration. Our research also includes *migration* as a process of transferring an application onto a cloud. Actually, we aim to present a sophisticated procedure to realize the transfer process to make the application run in the new cloud environment, which is realized manually by cloud experts.

Cloud application portability is a general ability to move applications between CSPs, no matter which cloud environment they are using as infrastructure. Cloud application portability is not considered as a data or service transfer between clouds, but as a transfer of a whole set of functionally organized services and data. Our goal is *automated cloud application portability* by defining an automated procedure that will realize migration or porting, without or with minimal user intervention. *Porting* is a complex process that usually needs a lot of support by cloud vendors, and especially by CSPs. Our goal is to use standard interfaces to cloud management, which will realize porting as a process. These cloud management features used by most of the existing operating systems (OS) and cloud environments integrate essential cloud management functions, such as invocation of virtual machines (VMs), instantiation, customization and management.

The approach for automated porting of a SOA application between different CSPs (or cloud environments) is based on the P-TOSCA model and practical implementation (Kostoska et al., 2014b). This model enables automated porting of Platform-as-a-Service (PaaS) hosted applications from one cloud environment to another as an extension of TOSCA 1.0 standard (OASIS, 2014).

Cloud portability requires the CSPs to enable cloud interoperability (Toosi et al., 2014). It means that a CSP must be able to replicate the application environment and enable application deployment.

Gusev M., Kostoska M., Ristov S. and Donevski A..
P-TOSCA Portability of SOA Applications.
DOI: 10.5220/0005407100710078
In *Proceedings of the 5th International Conference on Cloud Computing and Services Science* (CLOSER-2015), pages 71-78
ISBN: 978-989-758-104-5

2 RELATED WORK

As more CSPs and cloud environment vendors become available, users at some point would like to transfer their data and applications from one CSP to another, but there is no standard to do it seamlessly. A nice overview about cloud portability approaches and opportunities is given by Petcu and Vasilakos (2014). Gonidis et al. (2013) have classified three types of cloud portability solutions: 1) adoption of existing or emerging standards (like TOSCA, CDMI, OCCI, OCF), 2) usage of intermediary levels (like jClouds or mOSAIC) and 3) adoption of semantics and model-based solutions. Several approaches have been analyzed in (Ortiz Jr, 2011). IEEE P2301 is just one initiative to design a roadmap for application portability, management and interoperability interfaces, as well as for file formats and operating conventions.

Open Virtualization Format (OVF) (DMTF, 2010) establishes a transport mechanism for moving VMs from one hosted platform to another. It is an approach for definition of an open standard for packaging and distributing virtual appliances or more generally software to be run in multiple VMs. Their approach is based on using different hypervisors.

The OCCI, OVF and similar approaches work on IaaS level. When analyzed on PaaS level, the most promising approach is defined by TOSCA (OASIS, 2014) as a portable and manageable specification of services and applications deployed on any CSP. Our research with TOSCA specification was initially directed to test the feasibility of a TOSCA model deployment. Recently, analyzing this process we have identified critical points where original TOSCA specification requires further refinement and extended the specification with additional cloud-specific elements to enable automated porting (Kostoska et al., 2014b).

No commercial solution supports processing of TOSCA specification at this moment. (Binz et al., 2013) present the OpenTOSCA environment for imperative Cloud Service Archive (CSAR) processing. Unlike this proposal, P-TOSCA offers declarative processing and implementation for multiple cloud environments. Other initiatives include creation of visual environments for TOSCA specifications like Winery (Kopp et al., 2013) and Vino4TOSCA (Breitenbücher et al., 2012). Some approaches were concerned with creation of TOSCA specification for existing projects (Li et al., 2013; Kostoska et al., 2014a).

Petcu and Vasilakos (2014) also give an overview of tools and services that support a certain degree of portability, including Aoleus (cloud management software, written in Ruby), CompatibleOne (cloud broker, defining a language for management of cloud services), CloudFoundry (works on top of VMware-based IaaS), ConPaaS (federation support), Docker (deployment engine), mOSAIC (API that allows deployment and configuration management), Nimbus (a virtual site layer for dynamic provisioning of distributed resources), etc.

Katsratos et al. (2014) present a proof of concept for the portability problem on OpenStack cloud. They use Opscode Chef as a configuration management tool that describes and manages system configuration using a Ruby based domain-specific language. It automates the cloud management tasks that are obtained by translating a TOSCA-based application specification into Chef environment. Similar approaches are used by the existing EU funded research projects, such as SeaClouds, Remics, Cloud4SOA, Optimis, Contrail, Artist, PaaSage, MODAClouds, or even RighScale or CloudFoundry. Instead of building a TOSCA engine, these approaches translate a TOSCA-based application specification into a specification that can use cloud management tools, such as CAMP, Brooklyn, Chef, Puppet etc. However, our approach uses a practical implementation of P-TOSCA engine and direct application porting between clouds.

The concept of SOA services is based on unified communication and collaboration to produce the desired result (OASIS, 2014). It offers many benefits due to scalability and adaptability. The main SOA characteristics are Discoverable and Dynamically Bound, Self-Contained and Modular, Interoperability, Loose Coupling, Location Transparency, Composability, and Self-Healing (Valipour et al., 2009).

SOA applications consist of independent service elements orchestrated to communicate and exchange information in order to achieve the desired functionality. The services can be composed as applications (assembly of services and components bound by application logic), service federations (collections of services bound in large service domain) and service orchestration (execution of one business process by multiple successful service invocation) (Valipour et al., 2009).

Building and deploying a distributed SOA depends upon successful orchestration to enable services to be orchestrated in unified and defined process, successful deployment to enable proper configuration of security, reliability, scalability and successful management (Papazoglou and Heuvel, 2007).

3 P-TOSCA CONCEPTS

A P-TOSCA specification of an application contains XML description of the application topology (types,

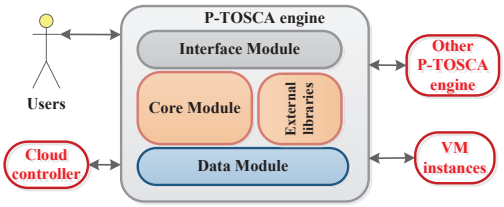

Figure 1: Context diagram of P-TOSCA engine.

templates and artifacts). This specification along with the required artifacts is packed in a CSAR. Artifacts represent files needed for application deployment or configuration like scripts, war files, zip files, libraries etc. The P-TOSCA approach is used to: *Migrate* an application onto a cloud; or *Port* an application from one CSP to another.

In our earlier work (Kostoska et al., 2014b), we have identified several TOSCA weaknesses and ambiguities and suggested extensions to enable a fully automated application life-cycle management. Most of these identified problems arise because the main goal of TOSCA is to be cloud, technology and hardware agnostic, while the real implementation needs proper definitions of several hardware based specific parameters, such as: a) Specifying the external namespace of ServerProperties; b) Specifying the initial number of instances child element to the ServerProperties element; c) Extending the ServerProperties element of Node Template with ServerIPAddress element; d) Specifying the external namespace of ScriptArtifact Properties; e) Extending the Properties element of Node Template with ServerSecurityProperties element; and f) Introducing XML defined plan.

All these extensions still keep the P-TOSCA cloud and technology agnostic. It just defines all those specific elements required for real implementation.

Details on the practical P-TOSCA engine implementation are also described in (Kostoska et al., 2014b). The software is hosted on a separate VM by the CSP and its architecture is presented in Fig. 1.

The interface module contains two parts, one to establish communication to the users and the other to collaborate with other P-TOSCA engine implementations. The core module is responsible for managing the CSAR archives. It executes the artifact plans and communicates with the cloud controller to manage, invoke and revoke various instances. All relevant data are stored and managed by the corresponding database module. Software is developed in Java programming language using Linux specifics.

The first use case, which presents the *cloud migration* (to migrate an application onto a cloud), is performed by a direct user interaction with a web application provided by the platform. The second use

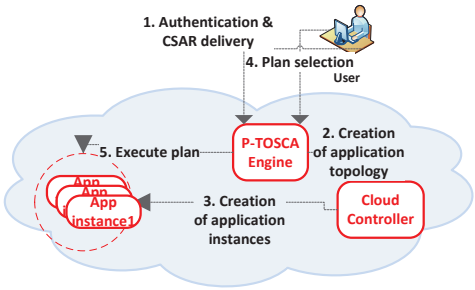

Figure 2: P-TOSCA based migration.

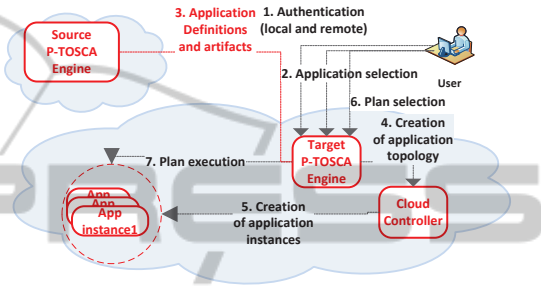

Figure 3: P-TOSCA porting.

case, which presents the *cloud application portability* (porting of an application from one CSP to another), is performed by a direct user interaction with a web application and communication between platforms using web services.

Both use cases may require only two clicks by the customer, one to select and upload the CSAR archive (or to select the appropriate application to be ported together with the P-TOSCA engine), and another for plan selection. We assume that the customer had already prepared the CSAR archive and the execution plan. As the plan selection may be a part of a script file, the complete transfer can be done automatically.

Fig. 2 presents a conceptual diagram of a sequence of activities to realize the cloud migration using the P-TOSCA platform. This use case includes the following actions: 1) A user authenticates at the P-TOSCA engine platform and uploads the CSAR archive; 2) The platform processes the archive and requires creation of instances according to the topology of the application; 3) The cloud controller creates the instances; 4) The user selects execution plans; 5) The platform executes the plans on the created instances.

The conceptual diagram describing the sequence of activities that realize interaction between the user and the P-TOSCA engine for cloud application portability is shown in Fig. 3. Porting an application from one CSP to another using P-TOSCA includes the following actions: 1) A user authenticates to the P-TOSCA engine, selects the remote P-TOSCA engine and authenticates to the remote platform; 2) The user

selects an application to be ported from the remote platform; 3) Application's definitions and artifacts are obtained from the remote platform via web services; 4) The platform requires creation of instances according to the specified application topology; 5) The cloud controller creates the instances; 6) The user selects execution plans; 7) The platform executes the plans on the created instances.

4 P-TOSCA DEMONSTRATION

A modified version of the eBay SOA Shopper project (Hansen, 2007) is used as a proof-of-concept of transferring a SOA application between clouds using the P-TOSCA model. The developed application uses the eBay SOAP API to retrieve offers from eBay by given search terms. It offers different interfaces (web service and application for web browser). It is a typical transaction-based application realized with the SOA approach that consumes services from one provider and offers services to other parties. This application is selected as SOA demo since it represents both a consumer and a service provider in same type (i.e. covers the two important aspects of SOA).

One Java EE container hosts the application. It consists of two main modules: *Interface module* that contains the services, which are offered as web interface, REST and SOAP services; *Core and consumer module*, which consumes services from eBay using the eBay SOAP API and converts the data in the required format. When an application user accesses an interface service (whether using SOAP, XML message or HTTP parameter), then the appropriate software module is invoked. The control then continues with the SOA Finder API (which represents a wrapper) with goal to invoke eBay services. Finally the result is returned to the user via corresponding interface. The application is hosted on one VM instance.

The application topology describes the application deployment architecture. The orchestration of application required elements is needed for: a) Enabling a platform independency; b) Easier installation; c) Cloud deployment. Orchestration, in this context, describes the way the services are invoked and managed.

TOSCA specifies the application topology nodes by node types divided in three main categories: *Base type*, which defines the basic components required by the application, such as OS, web server and web application; *Specific types* used to define the specific components required by the application topology, such as Linux OS, GlassFish web server and Java EE Web Application. The deployment and configuration of the application also requires usage of Maven and Ant tools. For that reason these tools are defined as specific types; *Custom types* that define the components developed specifically for the eBay Finder Application.

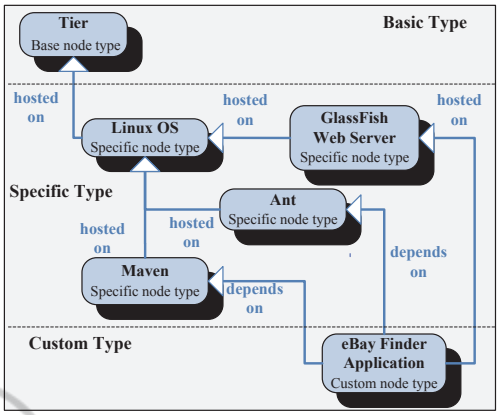

Figure 4: eBay Finder SOA application's topology template.

```
<NodeTemplate id="Tier"
    name="Instance for eBay Finder App"
    type="Tier">
      <Properties>
        <ServerProperties>
          <NumCpus>2</NumCpus>
          <Memory>2048</Memory>
          <Disk>10</Disk>
          <InitialNumInstances>
            1
          </InitialNumInstances>
          <ServerSecurityProperties>
            <ServerSecurityProperty>
              <protocol>TCP</protocol>
              <port>80</port>
            </ServerSecurityProperty>
            <ServerSecurityProperty>
              <protocol>TCP</protocol>
              <port>443</port>
            </ServerSecurityProperty>
          </ServerSecurityProperties>
        </ServerProperties>
      </Properties>
      ...
</NodeTemplate>
```

Listing 1: Tier node template definition.

The application topology template of the eBay Finder SOA application is presented in Fig. 4. Each node of the topology is specified by a node template, and the relationships between the nodes are specified using relationship template, depicted by blue color arrows. Same as nodes, the relationship types are initially set in the specification and each relationship template specifies the type of relationship. All elements are defined by XML. P-TOSCA model uses external namespaces for different custom defined elements, but they will be intentionally omitted from the further listings for clearer representation.

Tier node template definition is presented in Listing 1. In this context we use P-TOSCA extended specification of the *ServerProperties* element with the following: Initial number of instances to be

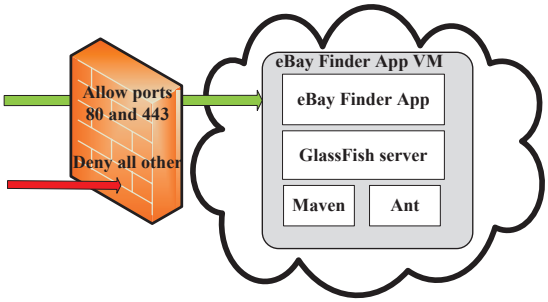

Figure 5: Communication layout of the eBay Finder.

```
<NodeTemplate id="GlassFishWebServer"
      name="GlassFish Web Server"
      type="GlassFishWebServer">
   <Properties>
      <httpport>80</httpport>
      <httpsport>443</httpsport>
      <username>admin</username>
      <password>adminadmin</password>
   </Properties>
</NodeTemplate>
```

Listing 2: *GlassFishWebServer* node template definition.

1; Server security properties with TCP protocol and port 80 (for HTTP); Server security properties with TCP protocol and port 443 (for HTTPS). The *ServerProperties* element was extended by definition of the *ServerSecurityProperty* element with corresponding port identification for HTTP and HTTPS protocols. It specifies details of the communication layout presented in Fig. 5.

Listing 2 shows definition of the GlassFish server node template. According to the P-TOSCA extended specification we define the *Properties* element with Port 80 (for HTTP) and Port 443 (for HTTPS). The user credentials are also defined within the *Properties* element.

Listing 3 shows the definition of the artifact template for configuration of the GlassFish server. The *ScriptArtifactProperties* element within the *Artifact Template* was extended with the *InputParameters* element used to define the input ports and user credentials. Note that these parameters correspond to the already defined properties of the GlassFish Web Server node template.

Unlike the standard TOSCA specification (that suggest usage of BPMN and BPEL languages), P-TOSCA uses XML defined plans. XML definition is used instead of use of BPMN and BPEL languages due to the ambiguity of these languages and the lack of BPMN processing engine at the moment when this project was developed (Kostoska et al., 2014b). Listing 4 shows the definition of the XML *Plan* element for application deployment.

Each *Plan* element contains the node templates and their operations for execution of the required action. The operations specified in the plan are executed sequentially in the order of description, unless some operation defines precondition (that should be executed before the operation).

```
<ArtifactTemplate id="glassfish-configsh" type="ScriptArtifact">
  <Properties>
    <ScriptArtifactProperties>
      <ScriptLanguage>sh</ScriptLanguage>
      <PrimaryScript>
        scripts/GlassFishWebServer/configure.sh
      </PrimaryScript>
      <InputParameters>
        <InputParameter
          nodeTemplateId="GlassFishWebServer"
          property="httpport"/>
        <InputParameter
          nodeTemplateId="GlassFishWebServer"
          property="httpsport"/>
        <InputParameter
          nodeTemplateId="GlassFishWebServer"
          property="username"/>
        <InputParameter
          nodeTemplateId="GlassFishWebServer"
          property="password"/>
      </InputParameters>
    </ScriptArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference
      reference="scripts/GlassFishWebServer">
      <Include pattern="configure.sh"/>
    </ArtifactReference>
  </ArtifactReferences>
</ArtifactTemplate>
```

Listing 3: GlassFish Artifact template.

```
<Plan id="InstallApplication"
    <NodeTemplateOperations>
      <NodeTemplateOperation ref="GlassFishWebServer">
          <Operation name="install"/>
          <Operation name="configure"/>
      </NodeTemplateOperation>
      <NodeTemplateOperation ref="Maven">
          <Operation name="install"/>
      </NodeTemplateOperation>
      <NodeTemplateOperation ref="Ant">
          <Operation name="install"/>
      </NodeTemplateOperation>
      <NodeTemplateOperation ref="eBayFinderApp">
          <Operation name="install"/>
          <Operation name="configure"/>
      </NodeTemplateOperation>
    </NodeTemplateOperations>
</Plan>
```

Listing 4: XML *Plan* element for eBay Finder deployment.

The first activity of the P-TOSCA portability sequence is the preparatory step, where the user authenticates and prepares the CSAR archive.

All definitions and the required artifacts are packed in the CSAR archive as a zip file. The CSAR archive for the eBay Finder SOA application contains: 1) Java EE 7 installation (which represents a zip file); 2) The application deployment artifact (war file); 3) Scripting artifacts for GlassFish; 4) Configuration, installation and deployment scripts. The final archive has a substantial size (over 90MB) due to the size of Java EE 7 installation file.

The next step includes copying to the P-TOSCA engine and starting a procedure defined by the corresponding migration or porting scenario. All these

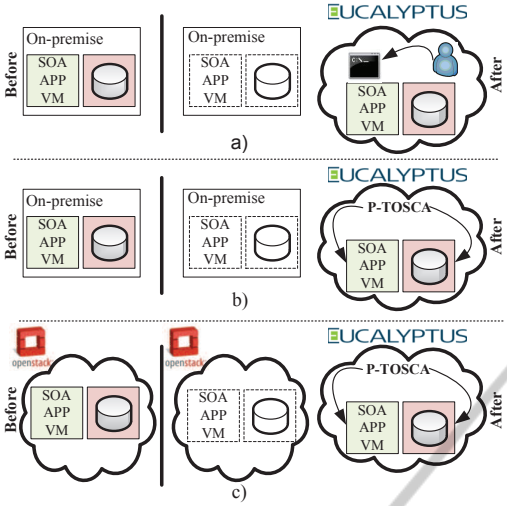

Figure 6: Three scenarios.

steps are performed in a sequence. The user interaction is required only to specify the execution plans.

5 TESTING METHODOLOGY

The test goal was to evaluate the functionality and deployment performance of the P-TOSCA portability model, and to provide a proof-of-concept of automated cloud application portability. A demonstration will be successful if the application migration and porting can be realized by using a P-TOSCA engine following the definitions of the P-TOSCA model. Performance evaluation will show if transferring the application using this approach is realized by a click of a button and consumes less time. Further on, we discuss details on the testing environment, test cases, and test data.

For testing purposes we used two cloud environments: OpenStack and Eucalyptus, isolated in separated VLANs. The OpenStack cloud was installed on one server with Ubuntu server 12.04 LTS. The server hosts all the OpenStack elements: node, cluster and tiers. The Eucalyptus cloud was installed on three physical servers with CentOS 6.5 in such a way that each server uses one Eucalyptus element.

Both frameworks were set with a P-TOSCA engine, which can communicate with the user and the cloud controller. Communication with other P-TOSCA engines is realized by web services.

Fig. 6 presents three different scenarios, where Eucalyptus is presented as the target cloud. *Manual migration onto a cloud (MM)* is a test scenario to evaluate the performance of application migration on the cloud, where the user manually creates an instance, deploys the application and sets authentication and security rules (Fig. 6 a); *P-TOSCA based migration onto a cloud (PTM)* is a test scenario to evaluate the performance of activities to upload the defined CSAR and to deploy the application using the web interface of the P-TOSCA engine (Fig. 6 b); *P-TOSCA based porting the application from one cloud to another (PTP)* is a test scenario to evaluate the performance to transfer a PaaS hosted application from one cloud environment to other using web services (Fig. 6 c).

The target cloud to migrate or transfer the application is either OpenStack or Eucalyptus cloud. Since both cloud frameworks are used for the three test scenarios, there are a total of 6 use cases.

Functionality testing is realized by testing the application after its deployment on the target cloud for its proper functioning. For this purpose we have selected several characteristic input parameters and specified the expected output. The functionality test actually realizes matching of the expected output and the real obtained output for the same input. Performance testing is defined by evaluation of the total time needed for deployment of the use case.

In the manual migration use case, the time measurement starts with user authentication and the activity of manual preparation, including collection of necessary files and packing into a zip archive, followed by the manual copying of the archive, manual extracting, and manual starting of the scripts, VM instantiation, execution plans and ends when the final installation and deployment activity is finished.

In the P-TOSCA based migration and porting, all activities are performed automatically by the corresponding scripts. The time is measured from the moment when the user authenticates, followed by starting the initial script and finishes with realization of the execution plan. In this use case we have defined user interaction for selection of an appropriate execution plan, although it can also be automated by a corresponding script.

6 EVALUATION

In addition to the original TOSCA specification, our P-TOSCA model gives details on implementation specifics, such as initial number of instances, used protocols and ports for communication, and relevant data about CPUs, memory, disks etc. This information is required for implementation and deploying purposes. XML specified plans and exiting P-TOSCA engine cannot support the full coverage of architecture configurations and deployments as those by the BPEL engine, but this is planned for next P-TOSCA

Table 1: Average time for use case scenarios.

Use case	id	sec	id	sec
MM	$T_d(O)$	1918	$T_d(E)$	1914
PTM	$T_m(O)$	1247	$T_m(E)$	1205
PTP	$T_p(O)$	1093	$T_p(E)$	1022

software release. Currently all sequential deployment and installation activities can be successfully applied by P-TOSCA.

Recently we have presented demo cases by using the P-TOSCA approach: a demo on a small SOA application (Ristov et al., 2014) and a demo of porting the N-tier applications (Gusev et al., 2014). Here we extend the approach on a more general SOA application and give comprehensive details on P-TOSCA application, along with functional and performance evaluation.

Proof of concept is demonstrated for all 6 use cases although OpenStack is not specifically designed neither for interoperability nor portability (Toosi et al., 2014). The overall application porting from Eucalyptus to OpenStack and vice verse are correspondingly demonstrated in the publicly available videos: http://youtu.be/92KaHt0CyxE and http://youtu.be/NRnqrPqO41k. Some performance related results are presented next to explore the influence of the cloud infrastructure.

Table 1 presents the average of 10 successful test case executions, calculated for manual, P-TOSCA based migration and for P-TOSCA based porting, where OpenStack and Eucalyptus are the target clouds, correspondingly. We observe similar results for both cloud platforms, that is, P-TOSCA based cloud migration is faster than the manual. P-TOSCA based porting is even faster, since the transfer is realized directly between the clouds. Let $c \in \{O,E\}$ identifies the cloud environment, where O stands for OpenStack and E for Eucalyptus. Denote by T_d the time for default manual migration (MM), $T_m(c)$ for P-TOSCA based migration (PTM), and $T_p(c)$ the time for P-TOSCA based porting (PTP).

The target cloud infrastructure is important for the overall deployment process. The obtained results show that migration and porting using P-TOSCA model is faster on Eucalyptus for 3.5% in case of migration and 6.95% in case of porting on Eucalyptus as target cloud instead of on OpenStack. This is due to the different cloud infrastructure used for hosting the cloud environment. The OpenStack cloud environment uses one server as infrastructure to host the node controller, cluster and tier, while the Eucalyptus uses three different physical servers as infrastructure. These results actually show that the better the infrastructure is, the faster the deployment.

The time needed for initial migration using P-TOSCA is slightly greater than the time for porting between the clouds because during the initial migration we access the platform from outer network. The reason is in the setting of the cloud environment and the duration of the copying process, which is due to the network latencies and throughput.

The experiment environment uses user interaction with WAN (Wide Area Network) protocol, while both clouds (OpenStack and Eucalyptus) are connected with 1Gb LAN (Local Area Network). So, the data transfer time for archive upload requires more time when using WAN, while the data transfer between clouds was faster due to the faster LAN protocol.

Although the size of the archive is greater than 90MB, most of the time differences between the manual and P-TOSCA migration are due to the time required by the user to access the visual interface of the cloud and to manually execute the actions.

Another time-consuming activity is introduction of the new cloud environment. In the conducted experiments we do not analyze the time needed for learning the environments (i.e. we assume them as known environment). In real life, the time needed for manual migration may be significantly longer if the users meet the new cloud environment for the first time. It takes time to get acquainted to the interface and the options offered by the CSP.

During this process the user does not have control over the instances (as in manual migration), since the platform uses specific security policies and authentication, but at the end of the process the user specific requirements are set.

Other SOA characteristic is modularity. If the SOA application consists of several loosely-coupled modules, each module can be defined as a separate node in the application topology and can be deployed on a dedicated instance, while at the same time the relationships between the modules can be described using the relationship template.

Our approach described on a PaaS level can be evaluated by the approach (Petcu and Vasilakos, 2014) with a high portability degree, since, no code has to be rewritten and recompiled, no restructuring of data and applications are required and no services are re-configured. All activities are realized straight forward and automatically, starting from archiving, transferring of archives, deployment and installation.

7 CONCLUSION

So far, TOSCA can be used with the BPEL engine, or extended to P-TOSCA and using the P-TOSCA en-

gine. All other published research results concern development of various tools that support the process, while the on-going projects translate the TOSCA definitions in specification used by specific cloud management tools. One of the benefits of the P-TOSCA platform is that the user does not have to learn and use the native interfaces of CSPs, making the management of hosting a SOA application an easy task.

A proof-of-concept of automated cloud application portability was demonstrated in this paper using P-TOSCA portability in case of migration or porting of a transaction-based SOA application.

The demonstration of migration and porting on OpenStack and Eucalyptus cloud environments can be used also for other cloud environments, since P-TOSCA uses a generalized approach for specification and modeling of an application, and provides a scripting mechanism for automated sequence of activities. The main benefit is the possibility to easily switch CSPs and port the application between clouds. It seems that this functionality is unattractive to CSPs, since they prefer vendor lock-in and would prefer not to give the customer an easy way out of their cloud. This looks similar to the ongoing fight for mobile phone devices by mobile providers. However, as the time goes by, the customers would prefer portability and easy way in and easy go out options from future CSPs. It is not a question of should CSPs do it, but when to do it.

REFERENCES

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA – a runtime for TOSCA-based cloud applications. *11th Int. Conf. on Service-Oriented Computing, LNCS* vol. 8274, pages 692–695. Springer.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Schumm, D. (2012). Vino4TOSCA: A visual notation for application topologies based on TOSCA. *OTM 2012, Part I, LNCS* vol. 7565, pages 416–424. Springer-Verlag.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616.

Distributed Management Task Force (2010). Open virtualization format specification version 1.1.0.

Erl, T. (2004). *Service-oriented architecture: Concepts, Technology, and Design*. Prentice Hall.

Gonidis, F., Simons, A. J., Paraskakis, I., and Kourtesis, D. (2013). Cloud application portability: an initial view. *6th Balkan Conf. in Informatics*, pages 275–282. ACM.

Gusev, M., Kostoska, M., and Ristov, S. (2014). Cloud P-TOSCA porting of N-tier applications. *22nd Int. TELFOR Forum, IEEE Conf. Publications*, pages 935–938.

Hansen, M. D. (2007). *SOA Using Java Web Services*. Pearson Education, Inc, Upper Saddle River, NJ.

Katsaros, G., Menzel, M., Lenk, A., Revelant, J. R., Skipp, R., and Eberhardt, J. (2014). Cloud application portability with TOSCA, Chef and Openstack. *Cloud Engineering (IC2E), 2014 IEEE Int. Conf.*, pages 295–302.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery – modeling tool for TOSCA-based cloud applications. *11th Int. Conf. on Service-Oriented Computing, LNCS* vol. 8274, pages 700–704. Springer.

Kostoska, M., Chorbev, I., and Gusev, M. (2014a). Creating portable TOSCA archive for iKnow university management system. *Federated Conf. Computer Science and Information Systems (FedCSIS), IEEE Conf. Publications*, pages 767–774.

Kostoska, M., Gusev, M., and Ristov, S. (2014b). P-TOSCA portability model for PaaS hosted applications. Tech. Report LiiT:22/2014, University Ss Cyril and Methodius, Computer Science and Engineering.

Li, F., Vogler, M., Claessens, M., and Dustdar, S. (2013). Towards automated IoT application deployment by a cloud-based approach. *Service-Oriented Computing and Applications (SOCA), 6th IEEE Int. Conf.*, pages 61–68.

OASIS (2014). Online files.

Ortiz Jr, S. (2011). The problem with cloud-computing standardization. *IEEE Computer*, 44(7):13–16.

Papazoglou, M. and Van Den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415.

Petcu, D. and Vasilakos, A. V. (2014). Portability in clouds: approaches and research opportunities. *Scalable Computing: Practice and Experience*, 15(3):251 – 270.

Rana, O. (2014). The costs of cloud migration. *Cloud Computing, IEEE*, 1(1):62–65.

Ristov, S., Kostoska, M., and Gusev, M. (2014). P-TOSCA portability demo case. *2014 IEEE 3rd Int. Conf. on Cloud Networking (CLOUDNET)*, pages 269–271.

Toosi, A. N., Calheiros, R. N., and Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, 47(1):7:1–7:47.

Valipour, M., Amirzafari, B., Maleki, K., and Daneshpour, N. (2009). A brief survey of software architecture concepts and service oriented architecture. *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE Int. Conf.*, pages 34–38.