The SePaDe System: Packaging Entire XaaS Layers for Automatically Deploying and Managing Applications

Kálmán Képes, Uwe Breitenbücher, Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany,
{kepes, breitenbuecher, leymann}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Kepes2017_SePaDe,
  author    = {K{\'a}lm{\'a}n K{\'e}pes and Uwe Breitenb{\"u}cher and Frank
               Leymann},
  title     = {The SePaDe System: Packaging Entire XaaS Layers for
               Automatically Deploying and Managing Applications},
  booktitle = {Proceedings of the 7th International Conference on Cloud
               Computing and Services Science, CLOSER 2017,
               24-26 April 2017, Porto, Portugal},
  year      = {2017},
  month     = apr,
  pages     = {626--635},
  isbn      = {978-989-758-243-1},
  publisher = {SciTePress}
}
```

The SePaDe System: Packaging Entire XaaS Layers for Automatically Deploying and Managing Applications

Kálmán Képes, Uwe Breitenbücher, and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{*kalman.kepes, uwe.breitenbuecher, frank.leymann*}*@iaas.uni-stuttgart.de*

Keywords: Application Deployment, Application Management, Automation, Portability, TOSCA

Abstract: The multitude of cloud providers and technologies diminish the interoperability and portability of applications by offering diverse and heterogeneous functionalities, APIs, and data models. Although there are integration technologies that provide uniform interfaces that wrap proprietary APIs, the differences regarding the services offered by providers, their functionality, and their management features are still major issues that impede portability. In this paper, we tackle these issues by introducing the SePaDe System, which is a pluggable deployment framework that abstracts from proprietary services, APIs, and data models in a new way: The system builds upon reusable archive templates that contain (i) a deployment model for a certain kind of application and (ii) all deployment and management logic required to provide defined functionalities and management features. Thus, by selecting appropriate templates, an application can be deployed on any infrastructure providing the specified features. We validate the practical feasibility of the approach by a prototypical implementation that is based on the TOSCA standard and present several case studies to evaluate its relevance.

1 INTRODUCTION

With the general acceptance of Cloud Computing, problems related to availability of IT resources seem to be diminished (Zhang et al., 2010). In Cloud Computing, accessing compute, network, and storage capabilities can be categorized according to the NIST (Mell and Grance, 2011) service models *Software-as-a-Service (SaaS)*, *Platform-as-a-Service (PaaS)*, and *Infrastructure-as-a-Service (IaaS)*: SaaS delivers complete software applications from within the cloud, while PaaS allows users to deploy their own applications on already available, scalable, and configurable hosting environments. IaaS allows a user to instantiate and access virtual resources, such as virtual machines and disks, which can be used to host the users' applications. While SaaS has the lowest management complexity, the complexity increases further from PaaS to IaaS as more and more management must be done by the user himself. For example, a user does not have to manage the scalability of an SaaS solution as this is done by the cloud provider's infrastructure, whereas he has to explicitly specify scaling rules when an application is hosted on an IaaS offering. Thus, depending on the users' requirements, he can select the most appropriate service model.

However, the multitude of vendors and their offerings diminish the interoperability and portability of applications with diverse and heterogeneous functionality, APIs, and data models. Although there are integration technologies that try to solve these issues on the PaaS and IaaS layer by abstracting from proprietary APIs and data models (Hoare, 2016), the differences regarding (i) the services offered by providers, (ii) their functionality, and (iii) their management features are becoming increasingly challenging. Indeed, such integration technologies help deploying applications on different clouds, but if required services, functionalities, or management features are not provided by the selected provider, applications cannot be deployed as desired. Thus, as the offerings of providers cannot be configured arbitrarily and as there are typically no natively-supported means to add required functionality or management features, this is a serious portability problem—from the functional as well as from the non-functional perspective.

In this paper, we tackle these issues. We introduce the *SePaDe System*, which is a pluggable deployment framework that abstracts from cloud providers and technologies in a new way: instead of providing unified interfaces that wrap proprietary APIs and normalizing data models, which are the typical approaches of integration technologies, the SePaDe System builds upon the concept of searching, packaging

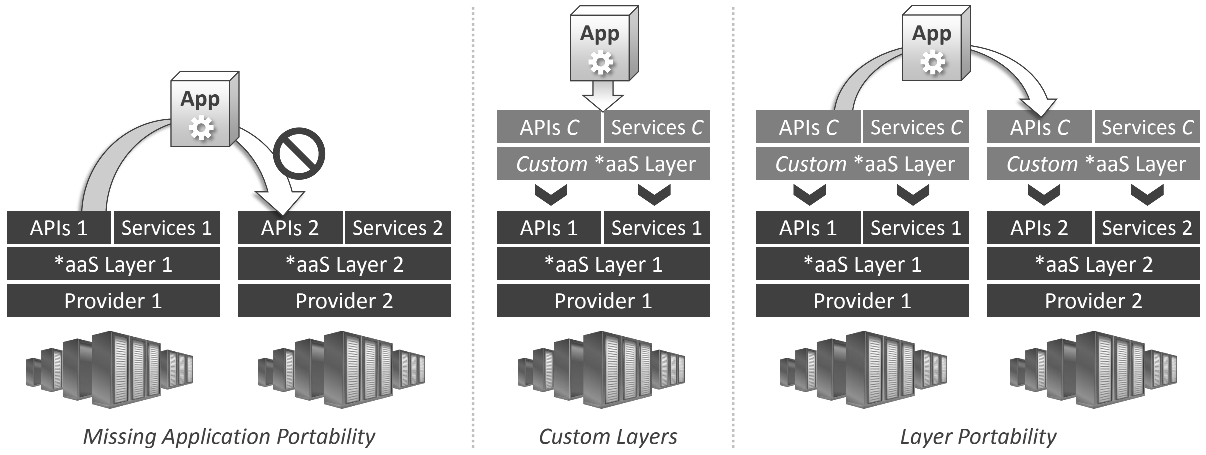

Figure 1: Overview of motivations for our work, ranging from issues of (i) missing application portability between different *aaS providers, (ii) the need for custom functionality on top of *aaS providers, and (iii) the need for *aaS layers to be portable.

and deploying *Self-Manageable Application Archive Templates (SMAART)*. SMAARTs are reusable archives that contain (i) a deployment model for a certain type of application on a certain infrastructure and (ii) all executables, e.g., shell scripts, required to deploy and manage the application to enable the functionalities and management features requested. The goal of the system is to enable the user to only specify the application files, the target infrastructure, and the desired features while the system automatically deploys the application following these requirements. We validate the practical feasibility by a prototype based on the TOSCA standard (OASIS, 2013): By applying SePaDe to TOSCA, we add an additional layer on top of TOSCA that enables users to search suitable templates that can be used to automatically deploy given application files following specified requirements. Moreover, we evaluate the practical relevance of our approach with case studies that show how different kinds of functionalities and features that are not natively supported can be added using the SePaDe System. Thus, besides increasing the degree of portability, the proposed concept can be also used to add new functionalities and features to existing offerings in a reusable way. Please note that our system does not replace existing technologies but adds another layer of abstraction.

The remainder of this paper is structured as follows: In Section 2, we motivate our approach based on the issue of portability. Afterwards, we present an overview of the SePaDe System, its architecture, and introduce the concept of SMAARTs in Section 3. We present a TOSCA-based prototype in Section 4 while Section 5 illustrates case studies. In Section 6 related work is discussed, Section 7 concludes the paper and outlines possible future work.

2 MOTIVATION & STATE-OF-THE-ART

In this section, we discuss three motivating scenarios that are difficult to realize using state-of-the-art technologies (see scenarios in Figure 1).

2.1 Missing Application Portability

The application portability problem results from vendors providing proprietary APIs, services, and data models (see left side in Figure 1). As a result, developing applications on top of one provider's APIs raises the issue of vendor lock-in, which impedes porting an application to another provider. Available integration technologies such as Nucleus (Röck and Kolb, 2016) tackle these issues by providing generic APIs and data models that wrap proprietary implementations. Also container technologies such as Docker (Fink, 2014) tackle portability issues. However, these approaches do not solve portability issues in general: if, for example, PaaS offerings of two different providers support deploying Docker containers, a container may be technically portable from a functional point of view. However, if one provider offers a required management feature, e.g., automated scaling, that is not supported by the other provider, the portability fails on the non-functional level. Thus, de facto and de jure standards and technologies only help partially regarding portability as they cannot support all variants that may be needed to get applications portable on the functional as well as on the non-functional level. The same applies to integration technologies: if one provider offers a service that is not supported by another provider, an application is not portable between these providers if this service is mandatorily required.

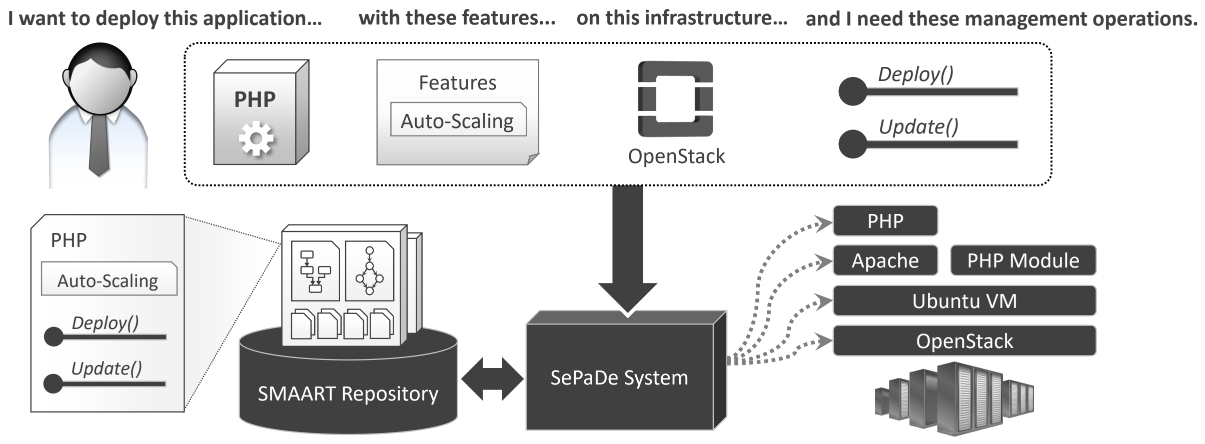

Figure 2: Overview of the SePaDe approach.

2.2 Custom Layers

Deploying custom layers on top of existing resources enables to use arbitrary hardware and eases the deployment of applications on these (see middle in Figure 1). Applications can use available resources in a transparent manner as they are developed on top of chosen platforms, and, therefore, are not bound to a specific environment. Especially within fields such as the Internet of Things (Atzori et al., 2010), it is not feasible to expect homogeneous resources. This trend raises the need for portable platforms which can be deployed on arbitrary resources. However, installing and configuring such layers is typically a complex, knowledge-intensive, and time-consuming task that requires automation. Our approach tackles this by enabling developers to build / reuse custom layers and to ship them in automatically installable archives. This enables to ship custom platforms which are highly specialized for a particular domain.

2.3 Layer Portability

Enabling developers to use their preferred layers decreases the issue of vendor lock-in, e.g., when an application is developed based on Cloud Foundry. By making the platform portable, the application can be migrated between different infrastructures, too (see right side in Figure 1). However, if a provider does not support all functions and features required by such a platform, the same issue of portability raises up one layer above. Nevertheless, bringing the whole platform logic to where it is needed is one concept of our approach. But in contrast to using a single portable platform for this purpose, we present a generic approach that is not coupled to a certain platform technology, which enables selecting an appropriate implementation for the target environment.

3 THE SEPADE SYSTEM

In this section, we introduce the *SePaDe System*, which is a pluggable deployment framework that abstracts from cloud providers and technologies in a completely new way. We first present an overview of the approach and detail the concepts in the following.

3.1 Overview of the Approach

The overall SePaDe approach is shown in Figure 2. The *SePaDe System* enables users to deploy the application automatically on a desired target infrastructure without the need to deal with technical deployment details. Thus, no technical expertise about deployment technologies such as Chef, virtualization technologies such as OpenStack, or cloud provider API handling is required. Using the SePaDe System, the user only has to specify the following information:

- *Application files* that shall be deployed as well as the application's type, for example, that the application is implemented in PHP.
- *Management features* that have to be provided automatically during operation, for example, that the application shall be automatically scaled.
- *Target infrastructure* on which the application shall be deployed, e.g., a certain OpenStack installation or a cloud provider such as Amazon.
- Required *Management operations* that can be triggered manually, for example, that an operation must be provided to update the application files.

The SePaDe System is connected to a *SMAART Repository* that contains reusable archive templates (*SMAARTs*) for deploying applications (see next subsection for details). Each of these SMAARTs can be used to deploy a certain type of application on a specific type of infrastructure and provides a defined set of management features and operations. As this information is annotated to each SMAART, the system can match this information with the specified user requirements and selects an appropriate SMAART for deploying the application files. All required deployment and management logic is contained in these SMAARTs, which provides a pluggable deployment approach as reusable archives for arbitrary target infrastructures and requirements that can be developed. Thus, the SePaDe System itself is a generic technology that only executes the deployment and management logic contained in SMAARTs and is, therefore, not bound to a certain technology. Using the system, users just have to upload their application files and to specify the desired infrastructure as well as the required functionalities and features, and the system selects an appropriate SMAART to automatically deploy the application following these requirements.

3.2 Self-Manageable Application Archive Templates (SMAARTs)

Our approach is based on application packages called *Self-Manageable Application Archive Templates (SMAARTs)*, which are reusable archive templates that contain (i) a deployment model for a certain type of application on a certain infrastructure, e.g., to deploy a PHP application on OpenStack. In addition, (ii) a SMAART contains generic executables, e.g., scripts, required to deploy and manage this type of application in a way that defined functionalities and management features are provided. The idea of SMAARTs is to put the user's application files into a suitable archive template that matches the user's requirements, which is then deployed automatically using a deployment system that is capable of invoking the contained deployment and management executables. We present a possible standards-based realization of this approach in Section 4. Thus, by selecting a suitable SMAART from the repository that fulfills the users requirements and putting the user's application files into it, arbitrary deployments on various infrastructures can be wrapped. This allows the reuse of available SMAARTs for a certain application type, a set of requested features, the specified target infrastructure, and required management operations. Moreover, the generic management executables in SMAARTs enable the automated management of the deployed application on runtime. These are either triggered automatically by the runtime or invoked by a user to execute a certain management operation.

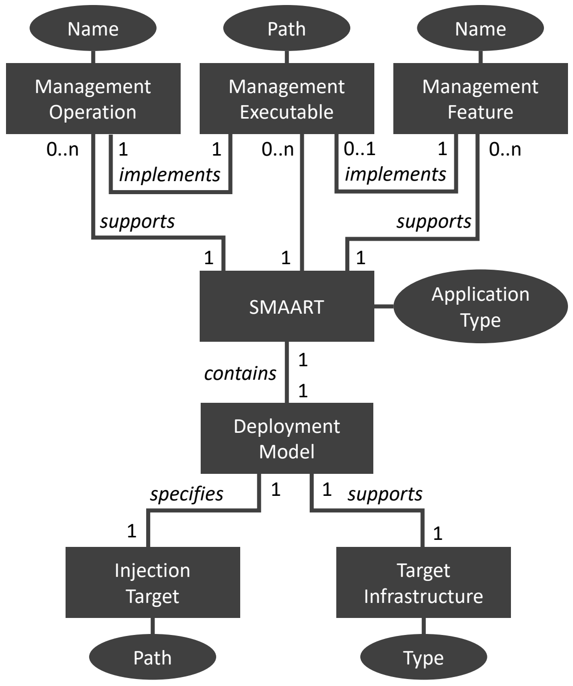

Figure 3: Meta-Model of Self-Manageable Application Archive Templates (SMAARTs).

The meta-model of SMAARTs as an ER model is shown in Figure 3. A SMAART has an *Application Type* property that specifies the type of applications it can deploy on a certain *Target Infrastructure*, which is specified by the *Deployment Model*. Moreover, this model describes all components and their relationships that are involved in this deployment. *Management operations* specify operations that can be explicitly invoked, e.g., to undeploy the application, while *Management Features* are built-in features that are enforced automatically, for example, automated scaling. The *Injection Target* is specified by the Deployment Model and defines the path in the archive template to which the application files must be copied. *Management Executables* implement operations or features and expect the application files exactly at this path.

To implement the deployment and management executables, the SMAART concept specifies no restrictions: Both, using low-level technologies such as scripts to implement these operations as well as using high-level orchestration approaches such as workflow technology may be used. Especially, workflow technology provides several standards such as BPEL (OASIS, 2007b) that is already used for automating the provisioning of applications (Breitenbücher et al., 2014). Of course, the employed deployment system has to support the chosen technology. We will present how to realize these concepts in Section 4 based on the OpenTOSCA ecosystem.

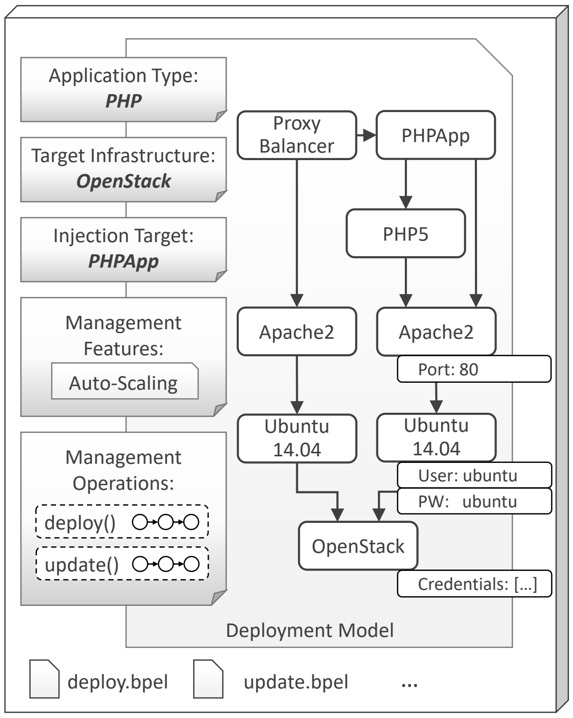

Figure 4: Example of a SMAART.

Figure 4 shows an example for the deployment of a PHP application on OpenStack, which is both specified by the *Application Type* and *Target Infrastructure*. The SMAART contains a deployment model and associated BPEL workflow that implements the `deploy()` operation: the *Management Executable deploy.bpel* installs the needed components such as the Apache2 HTTP server and the PHP modules on an Ubuntu virtual machine, which it provisions before on an OpenStack installation. The workflow extracts configurations, such as the HTTP port to be used, out of the deployment model. User-specific information, for example, the account information of the OpenStack to be used, are passed to the workflow via input parameters. Moreover, the SMAART contains an executable *update.bpel* that implements the operation to `update()` the deployed files. Both operations are exposed in the form of the *Management Operations* property. Thus, it defines supported management operations that can be triggered by the system or the user. Moreover, the SMAART also supports auto-scaling, which is specified by the *Management Features* property. In this example, installing and configuring auto-scaling is the responsibility of the deployment workflow. To enable the system to put the user's application files into the archive, the *Injection Target* specifies the path where the files need to be stored. The generic deployment and management workflows expect these application files exactly at this path.

3.3 SePaDe System & Matchmaking

In this section, we present the system architecture of the SePaDe system (see Figure 5). The main functionalities of the system are Selection, Packaging, and Deployment, hence *SePaDe*. In the following, we describe how the system enables the deployment of applications with a suitable SMAART.

Users specify their requirements in the user interface of the system. They enter (i) the application files to be deployed and their *Application Type*, e.g., a *PHP Application*, (ii) desired *Management Features* such as *auto-scaling*, (iii) the *Target Infrastructure*, e.g., a certain *OpenStack* installation, and finally (iv) the *Management Operations* they need, e.g., a `deploy()` and an `update()` operation. Of course, the user does not have to specify all of these requirements. For example, if the user only requests a special application type, features, and management operations but no special infrastructure, multiple SMAARTs that are based on different infrastructures are offered and the user selects one of them based on his preferences. When such a request arrives at the system, the *SMAART Selector* starts querying the *SMAART Repository* for a suitable SMAART that matches these inputs. To be a suitable SMAART, the following conditions must be fulfilled:

- The entered *Application Type* must match the *Application Type* of the candidate SMAART
- The entered *Target Infrastructure* must match the supported *Target Infrastructure* of the candidate
- The entered *Management Operations* must be a subset of the *Management Operations* supported by the candidate SMAART
- The entered *Management Features* must be a subset of the *Management Features* supported by the candidate SMAART

Based on these matchmaking rules, the SMAART Selector calculates a possible set of suitable SMAARTs that fulfill the requested criteria and selects afterwards one option from the set, e.g., based on different prices if SMAARTs are available for different providers. If no SMAART can be found, the application cannot be deployed in the requested way. After this selection step, the selected SMAART is passed to the *SMAART Packager* together with the given application files. This packager injects the passed application files to the path specified by the *Injection Target* property of the SMAART's *Deployment Model*. Please note, all *Management Executables* in the SMAART expect these files exactly there, thus, they can be executed automatically after this injection.

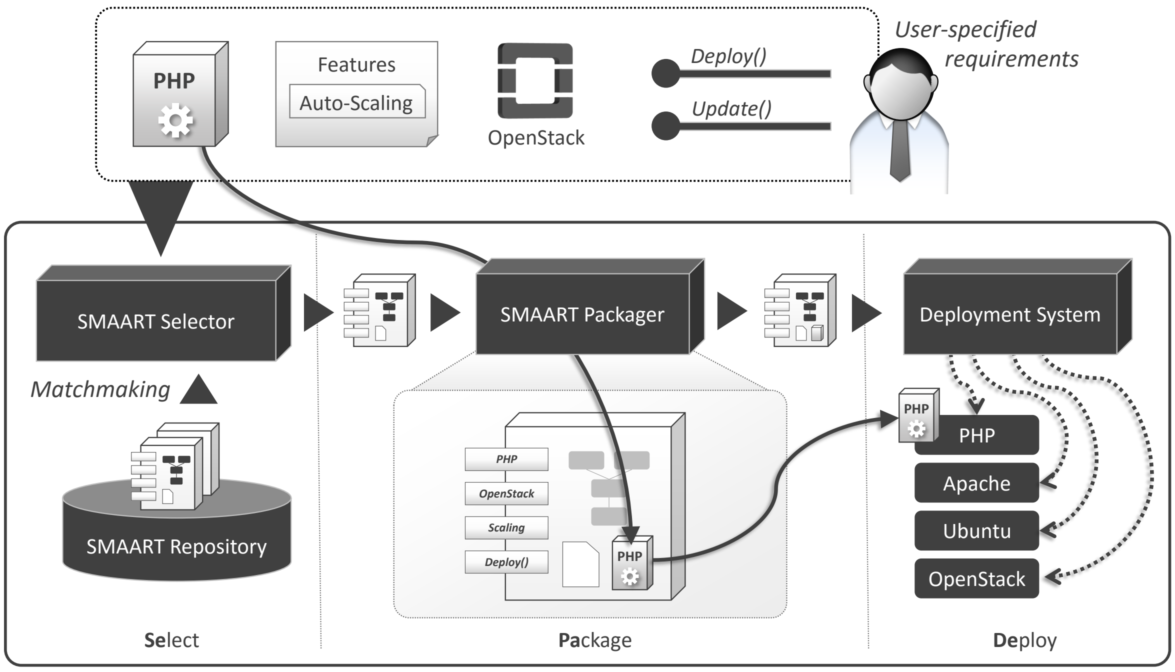

Figure 5: Overview of the SePaDe System Architecture.

Finally, the generated archive is shipped to a *Deployment System* that is capable of executing the *Management Executables* contained in the SMAART. To deploy an instance of this archive, i.e., to deploy the entered application files, the Deployment System invokes the *Management Executable* that implements the `deploy()` operation of the SMAART. Thus, this operation must be provided by each SMAART.

4 A TOSCA-BASED PROTOTYPE

In this section, we describe how we validated our approach using the TOSCA standard. We first describe the standard in Section 4.1, afterwards we describe how we mapped our SMAART meta-model to TOSCA in Section 4.2. In Section 4.3, we present our prototype that implements the SePaDe concept based on TOSCA CSARs and the open-source TOSCA environment OpenTOSCA (Binz et al., 2013).

4.1 An Introduction to TOSCA

In this section, we present an overview of the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* (OASIS, 2013), which enables automating the deployment and management of applications. TOSCA is an OASIS standard that specifies a meta-model for modelling the structure of cloud applications in the form of so-called *Topology Templates*. A Topology Template is a colored and attributed graph wherein nodes represent the components (*Node Templates*) of an application, and edges their relationships (*Relationship Templates*). Both Node and Relationship Templates are typed by *Node Types* or *Relationship Types*, respectively, specifying a template's semantics. Node Types expose *Management Operations* to manipulate instances of these types, e.g., a *VSphere* Node Type may expose operations to start and stop virtual machines. Thus, using these concepts various kinds of cloud infrastructures and technologies can be integrated as Node Types. Further, both types define a set of *Properties* that represent configurations of instances. A relationship between components can represent arbitrary types of relations specified by the corresponding Relationship Type, e.g., a *PHP Application* Node Template is *hostedOn* a *Apache 2* Node Template. Thus, as TOSCA's type system is extensible, arbitrary application structures of components and their relationships can be modelled.

TOSCA defines two types of artifacts: *Deployment Artifacts (DAs)* implement the business functionality of a Node Template, for example, a DA for a *PHP Application* Node Template are the PHP files implementing the application. The second type is called *Implementation Artifacts (IAs)*. IAs are used to implement the management operations defined by Node Types, e.g., . To orchestrate operations, TOSCA defines the concept of *Management Plans*, which are

```
1 <ServiceTemplate id="PHP_OpenStack" ...>
2  <Tags>
3   <Tag name="ApplicationType" value=
4       "{http://.../Types}PHP5Application"/>
5   <Tag name="InjectionTarget" value="DA12"/>
6   <Tag name="TargetInfrastructure" value=
7       "OpenStack-Liberty-12"/>
8   <Tag name="Features" value="AutoScaling"/>
9  </Tags>
10  <BoundaryDefinitions>
11   <Interfaces>
12    <Interface name="SePaDe-Lifecycle">
13     <Operation name="deploy">
14      <Plan planRef="BuildPlan"/>
15     </Operation>
16     <Operation name="update">
17      <Plan planRef="UpdatePlan"/>
18     ...
19  </BoundaryDefinitions>
20  <TopologyTemplate>
21    <NodeTemplates>
22     <NodeTemplate name="App" ...
```

Figure 6: TOSCA Service Template for SMAART.

executable workflow models that implement a certain management functionality, e.g., the provisioning of the entire application. All these elements are summarized in a *Service Template*. Moreover, TOSCA defines *Cloud Service Archives (CSARs)* to package all the mentioned elements into a self-contained archive file. Thus, TOSCA enables to embed all required software and models into such CSARs, which can be executed by standard-compliant runtimes to deploy and manage the application.

4.2 Mapping SePaDe to TOSCA

We map the SMAART meta-model (cf. Figure 3) to TOSCA as follows. SMAARTs are realized as TOSCA CSARs that contain a Service Template as shown in listing 6. For modelling the *Application Type*, *Management Features*, and *Target Infrastructure* supported by the SMAART, we used the concept of Tag-elements that can be attached to TOSCA Service Templates (lines 2-9). To realize the *Deployment Model* of the SMAART meta-model, we use TOSCA Topology Templates (lines 20-22). *Management Operations* supported by the SMAART are modelled using the Boundary Definitions of a Service Template, which contains interfaces that specify invokable operations (lines 10-19). *Management Executables* are realized as Implementation Artifacts that implement the operations exposed in the Boundary Definitions. The *Injection Target* is modelled as a Tag, whose value points to a Deployment Artifact's name to which the entered application files shall be attached. For attaching these files, only the referenced Deployment Artifact's file path property has to be updated. Using this injection mechanism, no Implementation Artifacts (i.e., *Management Executables*) are affected as these only refer to Deployment Artifacts by name and extract the file path property on runtime. Thus, the injection is seamless and does not affect executables. As a result, many concepts of SePaDe can be mapped to the native meta-model of TOSCA. Thus applying the SePaDes approach to TOSCA adds an additional layer on top of TOSCA (based on the SMAART meta-model) for searching suitable templates that can be used to automatically deploy application files following specified requirements.

4.3 Prototypical Implementation

In this section, we present our TOSCA-based prototype. We implemented the SePaDe concept into two systems: We extended (i) the open-source TOSCA modelling tool *Winery*[1] (Kopp et al., 2013) as well as (ii) the open-source TOSCA runtime environment *OpenTOSCA*[2] (Binz et al., 2013).

In the following, we describe the RESTful API which was implemented inside the OpenTOSCA Container to provide SePaDe's functionality. The API expects a single HTTP POST request with the following data: (i) the application file to be deployed, (ii) an XML QName indicating a TOSCA Artifact-Type which is the *Application Type*, (iii) a set of XML QNames denoting TOSCA Node Types that serves finer selection, (iv) a single XML QName denoting a TOSCA Node Type that serves as *Target Infrastructure*, (v) a set of strings stating the requested features for the application. For (i), we expect a single file which can range from archive files to binaries. This file is typed through the QName of the TOSCA Artifact Type specified by (ii). For (iii), we expect a possibly empty set of TOSCA Node Types specified as XML QNames which indicates that the SMAART to be selected must contain Node Templates of the given Node Types. This enables a finer-grained selection of SMAARTs. Our approach enables the deployment of applications on a target infrastructure and environment specified by (iv) with a single QName pointing to the desired target in the form of a Node Type. This Node Type specifies the infrastructure requested, e.g., OpenStack, Raspberry Pi, or Amazon AWS. As the last part, for (v) we expect an also possibly empty set of strings, which state requested *Management Features* of the SMAART. These strings are then checked against the tags defined inside TOSCA Service

[1]https://projects.eclipse.org/projects/soa.winery

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

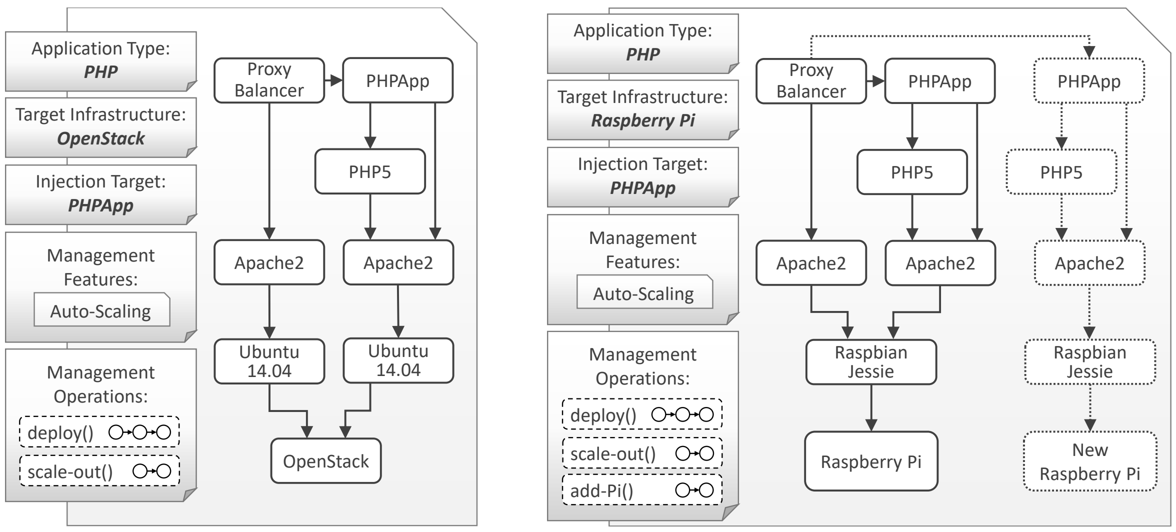

Figure 7: Case studies showing a custom layer for PHP apps and make the layer portable on OpenStack and Raspberry Pi

Templates implementing our SMAART meta-model. After sending such a request to the API, it will search inside the back-end for suitable SMAARTs, and after finding a suitable SMAART, the entered file is injected and the resulting CSAR is deployed on the OpenTOSCA container. To actually provision the application, i.e., to execute the `deploy()` operation, the user has to enter required input parameters into a user interface, e.g., credentials for OpenStack or the MAC-Addresses of the Raspberry Pis that have to be used. Then OpenTOSCA executes the associated executable for the `deploy()` operation, which is a BPEL or BPMN workflow model in our prototype.

5 CASE STUDIES

How our system complements existing integration technologies regarding the issue of *application portability* (Section 2.1) has been shown throughout the paper: by specifying non-functional requirements regarding management features, our approach is capable of providing them. For example, the SMAART (see Figure 4) used throughout the paper for deploying PHP applications on OpenStack shows how auto-scaling can be provided regardless of the actually used underlying system. To validate our concepts also regarding *custom layers* (Section 2.2) and *layer portability* (Section 2.3), we describe two case studies in this section with the goal to provision a single PHP application on different infrastructures with the means of packaging a custom layer in different SMAARTs.

The SMAART on the left in Figure 7 resembles our example SMAART from Figure 4, with the difference in available Management Operations. Here we altered the package to expose a `scale-out()` instead of an `update()` operation to show how we manage to scale applications on different infrastructures. While in the example version of the SMAART users can request an update of the applications' topology by executing `update()`, in this case the SMAART is able to scale the PHP application by executing the `scale-out()` operation. This is achieved with the TOSCA Plan implementing the `scale-out()` operation, which at runtime can instantiate a new PHP application stack on top of OpenStack, i.e., creating new instances of the *Ubuntu 14.04* virtual machine, the *Apache2* server, the *PHP5* module, and the *PHPApp* application. After creating such a new stack the `scale-out()` operation additionally adds the endpoint of the stack to the *Proxy Balancer* nodes' configuration to serve arriving request to it, enabling the overall application to serve more requests, thus scale. To show that our approach enables layer portability the second SMAART on the right in Figure 7 is also usable for PHP applications and implements the same Management Operations as the first (`deploy()` and `scale-out()`). Difference between the SMAARTs is the underlying infrastructure the PHP application will be hosted on, instead of an OpenStack cloud the right SMAART enables the usage of Raspberry Pis as the infrastructure. At the beginning of deploying such a SMAART all relevant components for the PHP application are installed, these components are analogous to the OpenStack case, but instead of installing the load balancer stack (*Apache2* and *Proxy Balancer*) and PHP application stack (*Apache2*, *PHP5* and *PHPApp*) on two distinct Ubuntu 14.04 instances, they are installed on the same initial Raspberry Pi at deployment time. After deployment the load balancer is listening on the intended application port, e.g., port 80, and redirects requests to the application. As a single Pi is not sufficient to keep up with large request loads, the package exposes the `add-Pi()` operation. Users can use this operation to add additional resources at runtime to the topology, which can then be used by the `scale-out()` operation to scale the overall application. The `add-Pi()` operation will when executed add a Raspberry Pi to the instance model, i.e, by creating a new Raspberry Pi node (*New Raspberry Pi*) and a node for its installed operating system (*Raspbian Jessie* on the right in the topology). Properties of the created nodes are taken from the input of the `add-Pi()` operation, containing information such as the MAC and/or IP address and credentials. When the system needs to scale, the `scale-out()` operation would detect the new Raspberry Pi nodes (hardware and os) and install a whole new stack of the PHP Application, i.e., installing an *Apache2*, a *PHP5* and a *PHPApp* node and add the new PHP application endpoint to the Proxy Balancers' configuration.

In summary, we showed how to extend available APIs with our approach to build *custom layers* by making management operations portable. Additionally, these case studies show *layer portability* by enabling the user to either deploy his PHP application on an OpenStack cloud or on a physical Raspberry Pi.

6 RELATED WORKS

Related works are concepts that introduce a layer on top of multiple PaaS providers, that reuse templates for cloud applications, and which allow users to develop applications without committing to a specific provider. In summary most approaches solve the challenges of PaaS platform interoperability by defining their own API on top of PaaS platforms. Hoare (Hoare, 2016) studied state-of-the-art research in PaaS interoperability and identified, beside the API approaches, the use of techniques in the field of the semantic Web. Loutas et al. (Loutas et al., 2011) introduce a PaaS Semantic Interoperability Framework that contains descriptions of different PaaS platforms to resolve interoperability issues. Yangui et al. (Yangui et al., 2013) describe a method on how to use the OASIS standard Service Component Architecture (SCA) (OASIS, 2007a) as a model for composite services. With these models a split into a set of services is taking place, which are afterwards deployed to so-called Micro-Containers which may already run inside one of the PaaS solutions. Yangui et al. argue that these Micro-Containers must be implemented once to be deployable into new PaaS environments, which in return allows the set of services to be deployed on those. Röck et al. (Röck and Kolb, 2016) tackle the problem of API differences between PaaS providers by introducing an abstract API on top. The general concept behind the API is to use adapters for different PaaS providers while the API itself exposes the capabilities of the environments as abstracted objects to the user, e.g., ranging from a generic *Service* to an *Application*. Similar to this is the Cloud4SOA platform by D'Andria et al. (D'Andria et al., 2012), which is also based on adapters for each supported PaaS platform. In contrast to our approach, these works introduce another API on top of the existing solutions, which nevertheless binds the applications against a particular interface. Our approach enables incorporating entire PaaS platforms inside a single SMAART, which allows to use the desired platform. It also enables installing platforms on bare metal, which additionally reduces portability issues. Yamato et al. (Yamato et al., 2014) introduce a Template Management Server which stores OpenStack HOT Templates for reuse by its users. The server enables user to create, share, extract from running OpenStack resources and update of templates to reflect changes in the environment. Additionally, the server is responsible for instantiating the templates by creating virtual resources inside the target OpenStack cloud. While this approach leverages a similar idea as our paper, it is limited to the OpenStack cloud, our approach can be implemented for arbitrary technologies as shown by the case studies. Another approach based around templates is presented by Gao et al. (Gao et al., 2012). Templates in their approach are virtual machine images that incorporate different software packages and services. Users can select an appropriate template from a repository and use it to provision their services. After such a request is made, the system selects appropriate cloud resources based on criteria given by the user to install the image and afterwards the services to run on those resources. While this approach is related to our method, it only deals with the deployment on virtual machines, our approach also incorporates bare-metal and arbitrary *aaS resources. Nguyen et al. (Nguyen et al., 2012) introduce the Blueprints meta-model, which specify a cloud application model according to basic properties, offerings, implementation artifacts, resource requirements, virtual architecture, etc. While a Blueprint allows specifying fine-grained information about a cloud application enabling to select an appropriate blueprint from a repository, these blueprints describe complete, deployable cloud applications and are not intended for reuse to deploy new applications.

7 CONCLUSIONS

In this paper, we presented an approach to deploy arbitrary applications on arbitrary infrastructures by leveraging the concept of templates. We showed in case studies how the reuse of deployment models can be enabled to deploy applications on heterogeneous resources such as virtual machines, PaaS layers, and also bare-metal machines. Further, our approach enables developers to choose their preferred platform and package their applications along with it in a single archive. This enables to port applications from one *aaS provider to another without the need to change implementations as the needed platform API's and services can be shipped together with the application.

Our approach enables wrapping entire *aaS layers in a reusable manner and, thereby, allows to cope with future challenges of growing fields such as IoT and Fog Computing. One challenge of those fields is to integrate local bare-metal resources with virtual resources of the cloud, and enable applications to be migrated between those resources. By enabling reuse of proven SMAARTs to automatically deploy *aaS layers on environments close to the "edge", the complexity of migrating applications from the traditional cloud to these scenarios is significantly reduced.

For future work, we plan to extend our approach by generating SMAART archives in an automated way, e.g., by determining the Application Type and Implementation Placeholder of a deployment model. Moreover, we plan to extend the approach to reference multiple infrastructures suitable to be used as underlying resources for the deployment model.

8 ACKNOWLEDGEMENTS

This work was funded by the BMWi project SmartOrchestra (01MD16001F).

REFERENCES

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer networks*, 54(15):2787–2805.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*, pages 692–695. Springer.

Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., and Wettinger, J. (2014). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *International Conference on Cloud Engineering (IC2E 2014)*, pages 87–96. IEEE.

D'Andria, F., Bocconi, S., Cruz, J. G., Ahtes, J., and Zeginis, D. (2012). Cloud4SOA: Multi-Cloud Application Management Across PaaS Offerings. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 407–414. IEEE, CPS.

Fink, J. (2014). Docker: a Software as a Service, Operating System-Level Virtualization Framework. *Code4Lib Journal*, 25.

Gao, W., Jin, H., Wu, S., Shi, X., and Yuan, J. (2012). Effectively deploying services on virtualization infrastructure. *Frontiers of Computer Science*, 6(4):398–408.

Hoare, S. (2016). A study of the state-of-the-art of PaaS Interoperability. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, page 7. ACM.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery – A Modeling Tool for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*, pages 700–704. Springer.

Loutas, N., Kamateri, E., and Tarabanis, K. (2011). A Semantic Interoperability Framework for Cloud Platform as a Service. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 280–287. IEEE.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing.

Nguyen, D. K., Lelli, F., Papazoglou, M. P., and Heuvel, W.-J. V. D. (2012). Blueprinting Approach in Support of Cloud Computing. *Future Internet*, 4(1):322–346.

OASIS (2007a). *SCA Assembly Model Specification Version 1.00*. Organization for the Advancement of Structured Information Standards (OASIS).

OASIS (2007b). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OASIS (2013). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS).

Röck, C. and Kolb, S. (2016). Nucleus – Unified Deployment and Management for Platform as a Service. *University of Bamberg, Tech. Rep.*

Yamato, Y., Muroi, M., Tanaka, K., and Uchimura, M. (2014). Development of template management technology for easy deployment of virtual resources on OpenStack. *Journal of Cloud Computing*, 3(1):7.

Yangui, S., Nasrallah, M. B., and Tata, S. (2013). PaaS-independent approach to provision appropriate cloud resources for SCA-based applications deployment. In *Semantics, Knowledge and Grids (SKG), 2013 Ninth International Conference on*, pages 14–21. IEEE.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18.