A Modelling Concept to Integrate Declarative and Imperative Cloud Application Provisioning Technologies

Uwe Breitenbücher[1], Tobias Binz[1], Oliver Kopp[1,2], Frank Leymann[1] and Johannes Wettinger[1]

[1]*IAAS, University of Stuttgart, Stuttgart, Germany*

[2]*IPVS, University of Stuttgart, Stuttgart, Germany*

Keywords: Cloud Application Provisioning, Automation, Declarative Modelling, Imperative Modelling.

Abstract: Efficient application provisioning is one of the most important issues in Cloud Computing today. For that purpose, various provisioning automation technologies have been developed that can be generally categorized into two different flavors: (i) declarative approaches are based on describing the desired goals whereas (ii) imperative approaches are used to describe explicit sequences of low-level tasks. Since modern Cloud-based business applications become more and more complex, employ a plethora of heterogeneous components and services that must be wired, and require complex configurations, the two kinds of technologies have to be integrated to model the provisioning of such applications. In this paper, we present a process modelling concept that enables the seamless integration of imperative and declarative provisioning models and their technologies while preserving the strengths of both flavors. We validate the technical feasibility of the approach by applying the concept to the workflow language BPEL and evaluate its features by several criteria.

1 INTRODUCTION

With the growing adoption of Cloud Computing in enterprises, the rapid and reliable provisioning of Cloud applications becomes a more and more important task. Especially the increasing number of available Cloud services offered by providers, e. g., Amazon and Google, provide powerful Cloud properties such as automatic elasticity, self-service, or pay-per-use features that are provided completely by the autonomous management capabilities of Cloud environments (Leymann, 2009). Due to this trend, more and more business applications are outsourced to the Cloud (Binz et al., 2014). As a result, Cloud-based business applications become (i) increasingly complex and (ii) employ a plethora of heterogeneous software, middleware, and XaaS components offered by different providers including non-trivial dependencies among each other. Consequently, the provisioning of such applications becomes a serious management challenge: (i) different kinds of Cloud offerings (IaaS, PaaS, SaaS, etc.) must be provisioned and (ii) complex configurations are required to setup and wire involved components. This typically requires the combination of multiple management technologies, especially if the application components are distributed across multiple Clouds (Breitenbücher et al., 2013).

However, combining (i) proprietary APIs, (ii) non-standardized configuration management tools, and (iii) different virtualization technologies in a single automated provisioning process is a complex modelling and integration challenge using traditional approaches such as workflows. The main reason for this complexity results from the nature of technologies that have to be combined: There are *declarative technologies*, such as Chef (Opscode, Inc., 2015; Nelson-Smith, 2013) or Puppet (Puppet Labs, Inc., 2015), which only describe the desired goal state of application components without specifying the actual tasks that have to be executed to reach this state. *Imperative technologies*, e. g., scripts or workflows, explicitly specify each technical step to be executed in detail. Although there are technologies for orchestrating imperative approaches with each other homogeneously (Kopp et al., 2012), combining declarative and imperative approaches results in implementing huge amounts of wrapper code as the two flavors are hardly interoperable with each other as there is no means to orchestrate them seamlessly.

In this paper, we tackle these issues. The first contribution is a detailed state of the art analysis of declarative and imperative provisioning approaches

Breitenbücher U., Binz T., Kopp O., Leymann F. and Wettinger J..
A Modelling Concept to Integrate Declarative and Imperative Cloud Application Provisioning Technologies.
DOI: 10.5220/0005495104870496
In *Proceedings of the 5th International Conference on Cloud Computing and Services Science* (CLOSER-2015), pages 487-496
ISBN: 978-989-758-104-5

including a critical evaluation. To tackle the analyzed issues, we present a modelling approach that enables integrating declarative and imperative provisioning models and the corresponding technologies seamlessly. We introduce the concept of *Declarative Provisioning Activities* that allows describing declarative goals directly in the control and data flow of an imperative workflow model. Based on our approach, developers are able to model provisioning workflows that specify not only imperative statements but declarative statements as well—without polluting the model with technical integration details. The approach enables to benefit from the strengths of both flavors: Declarative models can be used to specify high-level management goals, whereas imperative logic enables modelling complex cross-cutting configuration and wiring tasks on a very low level of technical abstraction. We validate the technical feasibility of the approach by presenting a prototypical implementation, which is integrated in the standards-based Cloud management ecosystem OpenTOSCA (Binz et al., 2013; Breitenbücher et al., 2014; Kopp et al., 2013) and the imperative workflow language BPEL (OASIS, 2007). We evaluate the approach by several criteria based on the conducted analysis and discuss its limitations.

The remainder of this paper is structured as follows: In Section 2, we conduct a detailed analysis regarding declarative and imperative provisioning technologies as well as combination concepts. Section 3 presents our approach of Declarative Provisioning Activities, which is validated in terms of a prototypical implementation in Section 5 and evaluated in Section 6. Section 7 concludes the paper and gives an outlook on future work.

2 STATE OF THE ART ANALYSIS

In this section, we conduct a detailed state of the art analysis of declarative and imperative provisioning approaches and existing technologies including a critical evaluation. Afterwards, we discuss related work that attempts to combine the two flavors.

2.1 The Declarative Flavor

Declarative approaches can be used to describe the provisioning of an application by modelling its desired goal state, which is enforced by a declarative provisioning system. They typically employ domain-specific languages (DSLs) (Günther et al., 2010) to describe goals in a declarative way, i. e., only the *what* is described without providing any details about the technical *how*. For example, a declarative specification may describe that a Webserver has to be installed on a virtual machine, but without specifying the technical tasks that have to be performed to reach this goal. The main strength of declarative approaches is that the technical provisioning logic, i. e., the technical tasks to be performed, is inferred automatically by the provisioning system, which eases modelling provisionings as the technical execution details are hidden (Herry et al., 2011). One of the most prominent examples of declarative provisioning description languages is Amazon CloudFormation[1]. This JSON-based language enables to describe the desired application deployment using Amazon's Cloud services including their configuration in a declarative model, which is consumed to fully automatically setup the application. In comparison to such provider-specific languages, which quickly lead to a vendor lock-in, provider-independent technologies were developed such as Puppet (Puppet Labs, Inc., 2015).

Due to the automatic inference of provisioning logic, declarative systems have to understand the declared statements. This restricts declarative provisioning capabilities to standard component types and predefined semantics that are known by the runtime (Breitenbücher et al., 2014). Thus, individual customizations for the provisioning of complex application structures cannot be realized arbitrarily and have to comply with the general, overall provisioning logic. As a consequence, the declarative approach is rather suited for applications that consist of common components and configurations, but is limited in terms of deploying big, complex business applications that require specific configurations with non-trivial component dependencies. Even mechanisms to integrate script executions, API calls, or service invocations at certain points in their deployment lifecycle, as supported by many declarative approaches, do often not provide the required flexibility as the overall logic cannot be changed arbitrarily. As the integration of other technologies is often not supported natively, models get polluted by glue and wrapper code, which results in complex models including low-level technical integration details (Wettinger et al., 2014). Nevertheless, the declarative flavor is very important due to (i) native support by Cloud providers and (ii) huge communities providing reusable artifacts.

2.2 The Imperative Flavor

In contrast to the declarative flavor, the imperative provisioning approach enables developers to specify each technical detail about the provisioning execution

[1] http://aws.amazon.com/cloudformation/

by creating an explicit process model that can be executed fully automatically by a runtime. Imperative models define (i) the control flow of activities, (ii) the data flow between them, as well as (iii) all technical details required to execute these activities. Thus, compared to declarative approaches, they describe not only *what* has to be done, but also *how* the provisioning tasks have to be executed. Imperative processes are typically implemented using (i) programming languages such as Java, (ii) scripting languages, e. g., Bash or Python, and (iii) workflow languages such as BPEL (OASIS, 2007) or BPMN (OMG, 2011). However, programming and scripting languages are not suited for the provisioning of serious business applications as they are not able to provide the robust and reliable execution features that are supported by the workflow technology (Leymann and Roller, 2000; Herry et al., 2011). Since general-purpose workflow languages do not natively support modeling features for application provisioning, we developed BPMN4TOSCA (Kopp et al., 2012), which is a BPMN extension that supports API calls, script-executions, and service invocations based on the TOSCA standard (OASIS, 2013) (a standard to describe Cloud applications). This language can be used to seamlessly integrate such tasks as it provides a separate activity-type for each of them. However, BPMN4TOSCA lacks support for the direct integration of declarative provisioning technologies, which need to be wrapped for their invocation. Thus, similar to general-purpose technologies, seamlessly integrating domain-specific technologies into one process is not possible. To wrap management technologies, we presented a management bus that provides a unified API for the invocation of arbitrary technologies (Wettinger et al., 2014). However, invoking the bus obfuscates the actual technical statements, which impedes maintaining and understanding process models.

Imperative approaches are suited to model complex provisionings that employ a plethora of heterogeneous components, especially for multi-cloud applications (Petcu, 2014). As they provide full control over the tasks to be executed, imperative models are able to automate exactly the manual steps that would be executed by a human administrator who provisions the application manually. Thus, while declarative approaches are rather suited for standard provisionings, imperative approaches enable developers to define arbitrary provisioning logic. The main drawback of the imperative approaches results from the huge amount of statements that must be modelled since the runtime infers no logic by itself. Consequently, manual process authoring is a labor-intensive, time-consuming, and error-prone task that requires a lot of low-level, technical expertise in different fields (Breitenbücher et al., 2014; Breitenbücher et al., 2013): Heterogeneous services need to be orchestrated (e. g., SOAP-based and RESTful provider APIs), low-level technologies must be integrated, and, especially, declarative technologies must be wrapped. As currently no technology supports the seamless integration of both flavors, their orchestration results in large, polluted, technically complex processes that require multiple different wrappers to support the various invocation mechanisms and protocols (Wettinger et al., 2014). These wrappers decrease the transparency as only simplified interfaces are exposed to the orchestrating process while the technical details, which are in many cases of vital importance to avoid errors when modelling multiple steps that depend on each other, are abstracted completely. In addition, wrappers significantly impede maintaining processes as not simply the orchestration process has to be adapted, but wrapper code needs to be modified and built again, too.

2.3 Integration Approaches

In this section, we present related work that attempts to combine both flavors. There are several general purpose concepts that attempt to bridge the gap between imperative provisioning logic and declarative models which generate provisioning workflows by analyzing the declarative specifications (Breitenbücher et al., 2014; Breitenbücher et al., 2013; Eilam et al., 2011; Keller et al., 2004; El Maghraoui et al., 2006; Herry et al., 2011; Levanti and Ranganathan, 2009; Mietzner, 2010). These approaches are able to interpret declarative specifications modelled using a domain-specific modelling language for generating provisioning plans, which can be executed fully automatically. The advantage of these approaches is the full control over the executed provisioning steps as the resulting workflows can be adapted and configured arbitrarily. However, the complexity, lack of transparency, and the polluted control and data flows of the resulting workflows are still problems that impede extending the plans if customization is required. Thus, the approach we present in this paper may be applied to these technologies for improving the quality of the generated processes. As a result of the discussion in this section, to ensure correct operation and to ease the creation of complex provisioning processes for non-trivial business applications, it is of vital importance to employ an extensible orchestration approach that supports the seamless orchestration of imperative and declarative provisioning technologies. Therefore, the main goals of this paper are (i) a seamless integration of declarative and im-

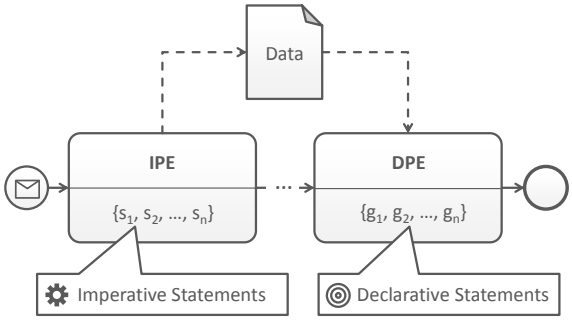

Figure 1: Concept of the direct integration approach.

perative provisioning modelling approaches as well as (ii) the orchestration of the corresponding provisioning technologies through workflow models.

3 INTEGRATED MODELLING

In this section, we present an approach that enables integrating declarative and imperative provisioning models seamlessly into the control and data flow of an imperative workflow. In Section 3.1, we introduce the abstract concept of the approach in a technology-independent manner and define data handling concepts in Section 3.2. In Section 4, we apply the approach to the workflow language BPEL in order to show how the concept can be realized using a concrete standardized workflow language.

3.1 Declarative Provisioning Activities

The general modelling approach is shown in Figure 1 and based on extending standardized, imperative workflow languages such as BPMN or BPEL by the concept of *Declarative Provisioning Activities*. These activities enable to specify declarative provisioning goals directly in the control flow of a workflow model that describes the tasks to provision a certain application. To present the conceptual contribution independently from a concrete workflow language, we first introduce the general concept in an abstract way and show its applicability to the standardized workflow language BPEL afterwards. Therefore, in this section, we distinguish only between (i) Imperative Provisioning Activities and (ii) Declarative Provisioning Activities that abstract from concrete realizations of provisioning tasks in different workflow languages. Of course, other control and data flow constructs, such as events and gateways, are also required to model executable processes. However, these are language-specific and do not influence the presented concept.

An *Imperative Provisioning Activity (IPA)* describes a technically detailed execution of a provisioning task as a sequence of one or more imperative statements. This can be, for example, a script implemented in Python or a simple HTTP-POST request that specifies a URL and data to be sent. Thus, the term is an abstraction of several existing imperative approaches such as scripts and programs that implement a workflow activity or the invocation of an API etc. The modeling and execution of such Imperative Provisioning Activities is supported natively by many workflow languages through general-purpose concepts or by domain-specific extensions, respectively. For example, BPMN natively supports the execution of script-tasks (OMG, 2011), the BPEL extension BPEL4REST (Haupt et al., 2014) enables sending arbitrary HTTP requests, and BPMN4TOSCA natively supports orchestrating provisioning operations based on the TOSCA-standard—especially the execution of configuration scripts on a target VM (Binz et al., 2013; Wettinger et al., 2014). This enables orchestrating arbitrary provisioning tasks using workflows that describe the technical details required for the automated provisioning of complex applications.

In contrast to this, we introduce the new concept of *Declarative Provisioning Activities (DPA)* in this paper that enables specifying desired provisioning goals in a declarative manner. A DPA consists of a set of declarative statements that describe *what* has to be achieved, e. g., a desired configuration of a certain application component, but without specifying any technical details about *how* to achieve the declared goals. Similar to other activity-constructs of workflow languages, Declarative Provisioning Activities are modelled directly in the control and data flow of the process model the same way as IPAs. This enables combining Imperative and Declarative Provisioning Activities intuitively while preserving a clear understanding about the overall flow. The operational semantics of Declarative Provisioning Activities are defined as follows: If the control flow reaches the activity, the declarative statements, i. e., the modelled goals, are enforced by the runtime that executes the workflow. The activity is executed until all goals are achieved and all affected application components are in the desired state specified by the DPA. Then, the activity completes and the control flow continues following the links to the next activities. Process models that contain both provisioning activity types are called *Integrated Provisioning Models* in this paper.

Figure 2 shows an example of an Integrated Provisioning Model that contains two Imperative Provisioning Activities and one Declarative Provisioning Activity, which (i) instantiate a virtual machine,

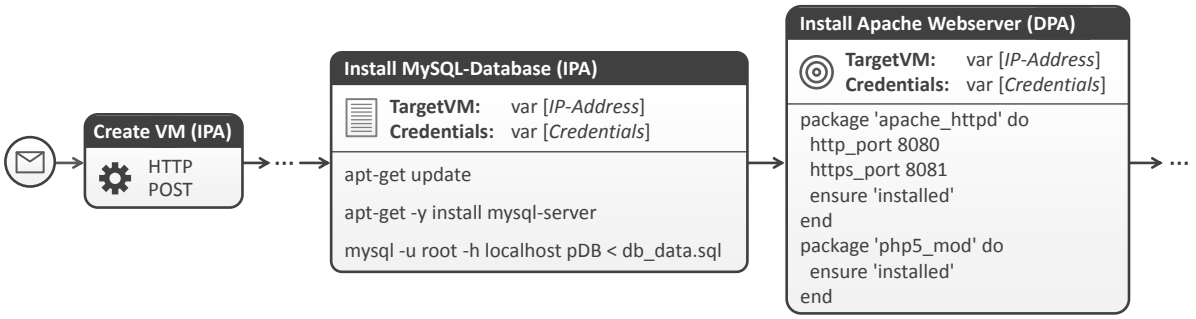

Figure 2: Simplified example of an Integrated Provisioning Model that (i) instantiates a virtual machine, (ii) installs a database, and (iii) installs a Webserver on the virtual machine (We omitted some tasks for reasons of space).

(ii) install a MySQL-database, and (iii) install an Apache Webserver on the virtual machine. The first IPA is an HTTP request to an API of a Cloud provider or an infrastructure virtualization technology that triggers the instantiation of the virtual machine. The activity specifies the request including all required configuration parameters and invokes the API correspondingly. After waiting for the successful instantiation, the IP-address and SSH credentials of the VM, which can be polled at the API, are stored in two variables of the workflow model: "IP-Address" and "Credentials". As these are standard tasks, we omit details in the figure for reasons of space.

The second IPA installs a MySQL database on the VM: The shown activity uses a low-level Bash script that imperatively specifies statements to be executed to install the database and to import a referenced SQL-file, which is uploaded to the VM by an IPA (omitted in the figure). To copy and execute this script on the VM, the process variables that store the IP-Address and SSH credentials of the target VM are used by the IPA to access the virtual machine via SSH and to execute the imperatively specified statements.

To model the installation and configuration of the Apache Webserver on the virtual machine, the DSL of the configuration management technology Chef (Opscode, Inc., 2015) is used to declaratively define the desired installation. Consequently, a Declarative Provisioning Activity is modelled that specifies the desired goals by declaratively describing the state and configuration of the Webserver that has to be enforced when executing the activity. Similarly to the second script-based IPA, the activity employs the same process variables to access the virtual machine.

This example shows that the direct integration of declarative and imperative languages and technologies in one orchestration process provides a powerful modeling approach as the corresponding imperative programming or scripting-languages, respectively, as well as the domain-specific languages of declarative approaches can be used seamlessly in one process model. Therefore, there is no need to write complex wrapper code or to invoke services wrapping these technologies that pollute the process model. Thus, the approach enables using the *right* technology for the *right* task while ensuring full-control over their orchestration without polluting the workflow model.

3.2 Data Handling

Both types of activities exchange data within the workflow. Therefore, we define three concepts including their operational semantics that enable describing the data flow between provisioning activities: (i) Input parameters, (ii) output parameters, and (iii) content injection. We continue abstracting from individual data storage concepts of workflow languages by simply referring to "process variables" and show in the next section how these concepts can be realized in the concrete workflow language BPEL.

As shown in Figure 2, the script-activity and the declarative Chef-activity reference process variables ("IP-Address" and "Credentials") that are assigned to a "TargetVM" and a "Credentials" attribute of the activities. These attributes represent predefined activity-specific *input parameters* of the activity implementation. When the activity gets executed by the workflow, the runtime copies the content of the referenced process variables "by value" and takes them as input parameters for invoking the implementation.

To exchange produced data between DPAs and IPAs, both may specify *output parameters* that contain the results of their execution. Each output parameter is represented as a pair of (i) *activity-internal data reference* and (ii) workflow process variable. An activity-internal data reference is a reference to a data container in the language of the activity. For example, an environment variable of a script. When the execution of the statements is finished, the referenced data is copied by the activity implementation to the specified process variables "by value".

Content injection enables using process variables

```
1  <extensionActivity>
2     <REST:POST ResponseVar="VMCreationResponse"
3                URL="https://ec2.amazonaws.com/?Action=RunInstances
4                            &ImageId=ami-31814f58
5                            &InstanceType=m1.small&..." />
6  </extensionActivity>
7     ...
8  <extensionActivity>
9     <DPA:Chef TargetVM="$bpelvar[IP-Address]" Credentials="$bpelvar[Credentials]">
10           package 'apache_httpd' do
11                  http_port $bpelvar[HTTPPort]
12                  https_port 8081
13                  ensure 'installed'
14           end ...
15    </DPA:Chef>
16 </extensionActivity>
```

Listing 1: Snippet of a BPEL model that employs an HTTP-Request as IPA and a DPA that declares Chef statements.

directly in the declarative or imperative language of a provisioning activity. These serve as placeholders that are replaced by the current content of the referenced variable when the execution of the activity starts. For example, a script may use the variable "IP-Address" to write the IP of the VM into firewall rules to enable accessing the Webserver from the outside.

4 REALIZATION USING BPEL

In this section, we prove that the presented approach is practically feasible by applying the integrated modelling concept to the workflow standard BPEL. Therefore, we (i) show how Imperative Provisioning Activities can be realized using existing constructs and extensions of BPEL and how (ii) Declarative Provisioning Activities can be modelled and executed using the so called "BPEL extension activities" (OASIS, 2007). The result is a *Standards-based Integrated Provisioning Modelling Language* that supports the direct orchestration of imperative languages as well as declarative languages.

In general, we realize DPAs by applying the BPEL concept of extension activities that allow to implement custom activity types in BPEL using programming languages such as Java (Kopp et al., 2011). BPEL-workflow runtimes support registering multiple different types of extension activities including their implementations. If the control flow of a workflow reaches an extension activity-element, its implementation is executed by the workflow engine and the whole XML-content of the extension activity-element in the BPEL model is passed to the implementation of the extension activity as input. Thus, the concept enables modelling arbitrary XML-definitions which are parsed and interpreted by the extension activity implementation. To select the right implementation, the element name of the extension activity-element's first child serves as lookup key for the workflow engine. Hence, we can realize arbitrary types of DPAs by implementing small programs that are executed when the control flow reaches one of these activities.

We show how IPAs und DPAs can be realized using extension activities by conducting an example. A modeller, e. g., developers or operations personnel (Hüttermann, 2012), manually models an Integrated Provisioning Model that consists of Declarative as well as Imperative Provisioning Activities where suitable. The XML shown in Listing 1 is an excerpt of a BPEL model that instantiates a virtual machine on the Cloud-offering Amazon EC2 and installs an Apache Webserver on it. The instantiation of the VM is modelled as activity that sends an HTTP-POST request to the management API of Amazon[2] (lines 1-6). We employ here the BPEL4REST extension activity approach (Haupt et al., 2014), which supports defining output parameters: The Amazon API synchronously returns the instance ID of the virtual machine in the HTTP response. As the provisioning of a virtual machine takes some time, the ID can be used to poll the status of the VM instantiation. Therefore, we store the response in a process variable called "VM-CreationResponse" (line 2). The implementation of the extension activity reads this mapping and writes the content of the HTTP response as value to the VM-CreationResponse variable. This variable can be used by other activities to monitor the current VM status

[2]http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_RunInstances.html

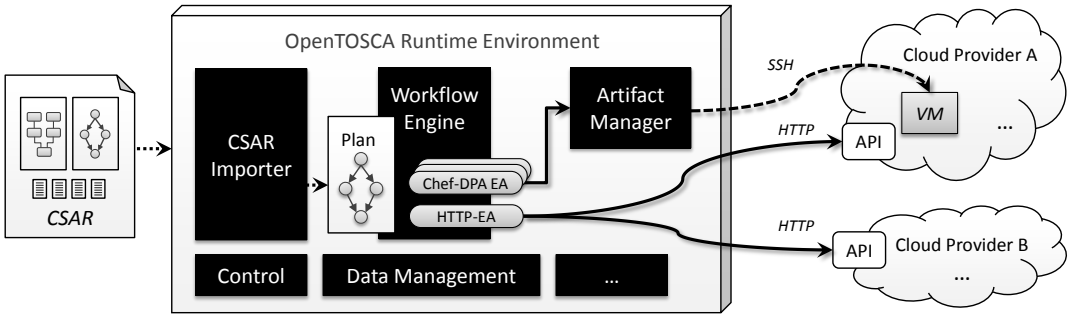

Figure 3: Prototypical implementation of the integrated concept based on the OpenTOSCA runtime environment.

and to retrieve the IP-address of the running virtual machine when the instantiation finished using similar API calls (omitted in Listing 1).

After the VM is provisioned, a Chef-DPA installs the Webserver on it (lines 8-16). This DPA defines the attributes used in our previous example with identical semantics. Similar to the HTTP extension activity, the extension activity implementation of the Chef-DPA reads its XML fragment, extracts the relevant information, and enforces the declared goals by accessing the VM using SSH, installing a Chef agent, and sending the declarative statements to this agent that enforces them. In this example, the input parameter concept is used to specify the target VM on which the Webserver has to be installed and the credentials to access the VM (line 9). The referenced BPEL variables of the workflow model are replaced by the extension activity implementation for execution. In addition, also the content injection concept is realized: In line 11, a BPEL variable is specified as configuration for the HTTP-port of the Webserver. Thus, when executing the DPA, its implementation retrieves the value of the "HTTPPort" workflow variable and replaces the placeholder before enforcing the declared configuration—similarly as for input parameters.

5 PROTOTYPE

To prove the technical feasibility of the presented approach, we implemented a prototype based on the OpenTOSCA ecosystem, which consists of the open-source modelling tool *Winery* (Kopp et al., 2013), the imperative runtime environment *OpenTOSCA* (Binz et al., 2013), and the self-service portal *Vinothek* (Breitenbücher et al., 2014). The system is based on the TOSCA standard (OASIS, 2013), which enables developing application packages that are portable across different platforms. TOSCA specifies a metamodel for (i) describing the application's structure as topology model and (ii) enables using management workflows to provision and manage the modelled applications. In addition, TOSCA standardizes a package format called *CSAR* that contains the topology model, all management workflows, and all artifacts that are required to provision and manage the described application, e. g., application files or installation scripts. CSARs can be created using the modelling tool Winery. A CSAR is consumed by the OpenTOSCA runtime which deploys the workflows contained therein. Therefore, the runtime employs a workflow engine (WSO2 BPS)[3] to execute BPEL workflows. Using the Vinothek, their execution can be triggered.

Figure 3 shows a simplified architecture of the OpenTOSCA runtime environment including our prototypical realization of the integrated modelling concept. The *CSAR Importer* is responsible for consuming CSARs and processing the contained data, e. g., by storing the models in local databases. The *Control* then triggers the local deployment of all management workflows so that they can be executed by the Vinothek to provision a new application instance or to manage a running instance. The concept presented in this paper is realized by implementing extension activity-plugins for the workflow engine. The HTTP-extension activity, for example, can be used in BPEL workflows to invoke management APIs of providers to instantiate or manage virtual machines. As described in the previous section, DPAs can then use process variables to access these virtual machines in order to install or configure software etc. To implement these extension activities, e. g., the Chef-DPA, we delegate executing the declaratively described goals to a component called *Artifact Manager*. This plugin-based manager is able to execute various configuration management technologies such as Chef or also imperative scripts, e. g., Bash scripts (Wettinger et al., 2014). Thus, implementing IPAs and DPAs is eased by invoking this manager. Of course, arbitrary technologies can be integrated without using the manager, too. For modelling Integrated Provi-

[3] http://wso2.com/products/

Table 1: Criteria Evaluation.

Feature	Declarative	Imperative	Integrated Approach
Full control		x	x
Complex deployments	(x)	(x)	x
Hybrid and multi-Cloud applications	(x)	x	x
Seamless integration			x
Component wiring	(x)	x	x
XaaS integration	(x)	x	x
Full automation	x	x	x
Straightforwardness	x		x
Extensibility	(x)	x	x
Flexibility		(x)	(x)

sioning Models, we employ the modelling tool *BPEL Designer*[4]. Since the prototype is based on TOSCA and BPEL, it provides an end-to-end, standards-based Cloud application management platform that enables integrating various technologies seamlessly.

6 EVALUATION

In this section, we evaluate the presented approach by comparing it with the plain declarative and imperative management flavors. For the comparison, we reuse the management feature criteria for comparing service-centric and script-centric management technologies (Breitenbücher et al., 2013) and additionally add criteria that are derived from the features of each flavor discussed in Section 2. As a result, the criteria represent requirements that must be fulfilled to fully automatically provision the kind of complex composite Cloud applications described in the introduction (cf. Section 1). An "x" in Table 1 denotes that the corresponding approach fully supports the criterion. An "x" in parentheses denotes partial support.

Full control means that provisioning may be customized arbitrarily by the process modeller in each technical detail. As declarative approaches infer the details about the execution by themselves, the general provisioning logic cannot be changed easily. In contrast to this, imperative approaches explicitly model each step to be performed and can be, therefore, customized arbitrarily. Because the integrated approach supports both, it fulfills this criterion completely.

Complex deployments denotes that real, non-trivial business applications that employ various heterogeneous components and services can be deployed using a technology of the flavor. Declarative approaches reach their limits at a certain point of required customizability: as the provisioning logic is inferred by a general-purpose provisioning system, only known declarative statements can be understood and processed (cf. Section 2). Thus, if a very specific, arbitrarily customized application structure or configuration has to be deployed, declarative approaches are often not able to fulfill these rare and very special requirements completely. The integration of low-level execution code such as scripts partially solves this problem. In contrast to this, based on the full control criterion, in general arbitrary complex provisionings can be described using imperative approaches such as scripts or workflows. However, the technical complexity of the resulting processes is often hardly manageable and maintainable as the integration of technologies, as explained in Section 2, leads to a lot of glue and wrapper code, which results in many lines of process implementation code. Thus, plain imperative approaches are not ideal for handling such cases completely and are, therefore, only partially suited. The integration approach presented in this paper solves these issues as the optimal technology can be chosen without polluting the process with wrapper code.

The *hybrid and multi-Cloud applications* criteria evaluate the support for applications that are either hosted on (i) a combination of private and public Cloud services or (ii) Cloud services offered by different providers. Since many declarative approaches such as Amazon CloudFormation employ proprietary, non-standardized domain-specific languages, many of these technologies are not able to provision a distributed application as described above. General purpose technologies such as TOSCA (OASIS, 2013) allow to provision hybrid as well as multi-Cloud applications, for example, by using the TOSCA plan generator (Breitenbücher et al., 2014). However, if multiple providers are involved, typically their proprietary languages have to be used as the declarative general-purpose technologies are not able to support all individual technical features. Based on the criteria *full control* and *complex deployments*, the imperative as well as the proposed approach fulfill this criterion.

Seamless integration evaluates the capability to

[4]https://eclipse.org/bpel/

employ arbitrary management technologies without (i) polluting the model or (ii) leading to abstracted wrapper calls (cf. Section 2). As extensively discussed in the previous sections, neither declarative nor imperative approaches natively support all required integration concepts. In contrast, the presented approach fulfills this criterion due to the introduced concept of Declarative Provisioning Activities.

The *component wiring* criterion means that multiple application components can be wired. Declarative approaches support this partially as unknown components or complex wiring tasks cannot be described in an arbitrary manner. The imperative as well as the integrated approach solve this issue as any task to wire such components can be orchestrated arbitrarily.

XaaS integration means the ability to orchestrate various kinds of Cloud services that represent application components. Generic declarative approaches support this only partially as complex configuration tasks are hard to model. Proprietary approaches such as Amazon CloudFormation are bound to a certain provider and, therefore, require glue code to integrate other services. The imperative and the presented approach fully support this requirement following the argumentation of *component wiring*.

The *full automation* criterion is fulfilled by all kinds of approaches, as all of them enable a fully automated provisioning of the described applications.

Straightforwardness evaluates, if describing the provisioning of an application can be done in an efficient manner requiring appropriate effort. The declarative approaches are typically easy to learn, as technical complexity is shifted to the provisioning systems and only the desired goals have to be specified. Imperative approaches such as scripts or workflows quickly become huge and complex due to the directly visible low-level details about the (i) control flow and the (ii) data flow. In addition, in many cases, trivial steps have to be modelled explicitly. The presented integration approach fulfills this criterion completely as the optimal technology can be selected for a certain provisioning task. Even a single DPA may be modelled that declares all provisioning goals.

The *extensibility* criterion means the ability to involve other management technologies. Declarative approaches allow this by using glue code at certain points in the inferred logic. Due to the *full control* criterion, imperative approaches are able to include arbitrary implementations at any point in the process. Thus, the integrated approach supports this feature.

The declarative approaches do not support *flexibility* due to the *full control* criterion. However, also using imperative approaches are limited in terms of flexibility: If a complex application leads to a huge provisioning process, adapting this process is a challenging task. Therefore, imperative as well as the presented approach fulfill this criterion only partially. To tackle these issues, we conduct research on modelling situation-aware processes to increase the flexibility.

To summarize the evaluation, the presented approach profits from all benefits of the two provisioning flavors while solving drawbacks by the strengths of each other. Whereas complex application provisionings can be modelled in a flexible manner preserving the full control over the provisioning, standard tasks can be modelled easily using declarative specifications in a straightforward manner. Even distributed application structures, for example, hybrid and multi-Cloud applications can be provisioned using the integrated approach described in this paper. One of the most important criterion, the seamless integration of provisioning technologies, is solved by the concept of Declarative Provisioning Activities while imperative technologies are typically integrated already in existing languages. Thus, while the resulting process models are implemented in a standards-compliant manner, intuitive provisioning modelling helps developing and maintaining models.

6.1 Limitations

In this section, we discuss the limitations of the presented approach. A drawback is the tight coupling of Integrated Provisioning Models to the structure of the application to be provisioned. Imperative orchestrations to provision the components of a certain application structure are sensitive to structural changes: Different combinations of components lead to different models that must be created and maintained separately (Breitenbücher et al., 2013; Eilam et al., 2011; El Maghraoui et al., 2006). Thus, as the concept of Integrated Provisioning Models is based on imperatively orchestrating the two kinds of provisioning activities, this applies also for the approach presented in this paper. As a result, Integrated Provisioning Models for new applications often have to be created from scratch while maintaining existing processes results in complex, time-consuming adaptations (Breitenbücher et al., 2014). To tackle this tight coupling of imperative orchestrations and concrete structures, we did research on generic process fragments for application management, which can be reused for individual applications (Breitenbücher et al., 2013; Breitenbücher et al., 2013). We plan to combine this approach with the presented concept. In addition, currently only the provisioning of applications is supported by our concept. Therefore, we plan to extend the concept to support also management.

7 CONCLUSION

In this paper, we presented a process modelling approach that enables the seamless integration of imperative and declarative provisioning models by introducing the concepts of (i) Declarative Provisioning Activities and (ii) Integrated Provisioning Models. The approach enables intuitive provisioning modelling without handling technical integration issues of regarding different technologies and domain-specific languages that pollute the control as well as the data flow of the resulting workflow models. To prove the technical feasibility of the approach, we applied the presented concept to the workflow language BPEL and extended the standards-based application management system OpenTOSCA. In addition, we evaluated its features by several criteria. The evaluation shows that the presented approach enables to benefit from strengths of both flavors. In future work, we plan to apply the concept also for application management.

ACKNOWLEDGEMENTS

This work was partially funded by the projects SitOPT (Research Grant 610872, DFG) and NEMAR (Research Grant 03ET40188, BMWi).

REFERENCES

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). Migration of enterprise applications to the cloud. *it - Information Technology, Special Issue: Architecture of Web Application*, 56(3):106–111.

Binz, T. et al. (2013). OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *ICSOC 2013*, pages 692–695. Springer.

Breitenbücher, U., Binz, T., Kopp, O., and Leymann, F. (2013). Pattern-based runtime management of composite cloud applications. In *CLOSER 2013*, pages 475–482. SciTePress.

Breitenbücher, U., Binz, T., Kopp, O., and Leymann, F. (2014). Vinothek - A Self-Service Portal for TOSCA. In *ZEUS 2014*, volume 1140 of *CEUR Workshop Proceedings*, pages 69–72. CEUR-WS.org.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2013). Integrated cloud application provisioning: Interconnecting service-centric and script-centric management technologies. In *CoopIS 2013*, pages 130–148. Springer.

Breitenbücher, U. et al. (2014). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *IC2E 2014*, pages 87–96. IEEE.

Eilam, T., Elder, M., Konstantinou, A., and Snible, E. (2011). Pattern-based composite application deployment. In *IM 2011*, pages 217–224. IEEE.

El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., and Konstantinou, A. V. (2006). Model driven provisioning: bridging the gap between declarative object models and procedural provisioning tools. In *Middleware 2006*, pages 404–423. Springer.

Günther, S., Haupt, M., and Splieth, M. (2010). Utilizing Internal Domain-Specific Languages for Deployment and Maintenance of IT Infrastructures. Technical report, Very Large Business Applications Lab Magdeburg, Otto von Guericke University Magdeburg.

Haupt, F., Fischer, M., Karastoyanova, D., Leymann, F., and Vukojevic-Haupt, K. (2014). Service Composition for REST. In *EDOC 2014*. IEEE.

Herry, H., Anderson, P., and Wickler, G. (2011). Automated planning for configuration changes. In *LISA 2011*. USENIX.

Hüttermann, M. (2012). *DevOps for Developers*. Apress.

Keller, A., Hellerstein, J. L., Wolf, J. L., Wu, K. L., and Krishnan, V. (2004). The champs system: change management with planning and scheduling. *Network Operations and Management Symposium, 2004*, pages 395–408.

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

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery – A Modeling Tool for TOSCA-based Cloud Applications. In *ICSOC 2013*, pages 700–704. Springer.

Kopp, O. et al. (2011). A Classification of BPEL Extensions. *Journal of Systems Integration*, 2(4):2–28.

Levanti, K. and Ranganathan, A. (2009). Planning-based configuration and management of distributed systems. In *IM 2009*, pages 65–72.

Leymann, F. (2009). Cloud Computing: The Next Revolution in IT. In *Proc. 52th Photogrammetric Week*, pages 3–12.

Leymann, F. and Roller, D. (2000). *Production workflow: concepts and techniques*. Prentice Hall PTR.

Mietzner, R. (2010). *A method and implementation to define and provision variable composite applications, and its usage in cloud computing*. Dissertation, University of Stuttgart, Germany.

Nelson-Smith, S. (2013). *Test-Driven Infrastructure with Chef*. O'Reilly Media, Inc.

OASIS (2007). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. OASIS.

OASIS (2013). *Topology and Orchestration Specification for Cloud Applications Version 1.0*.

OMG (2011). Business Process Model and Notation (BPMN), Version 2.0.

Opscode, Inc. (2015). Chef official site: http://www.opscode.com/chef.

Petcu, D. (2014). Consuming resources and services from multiple clouds. *Journal of Grid Computing*, 12(2):321–345.

Puppet Labs, Inc. (2015). Puppet official site: http://puppetlabs.com/puppet/what-is-puppet.

Wettinger, J. et al. (2014). Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA. In *CLOSER 2014*, pages 559–568. SciTePress.