Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications

Christian Endres[1], Uwe Breitenbücher[1], Michael Falkenthal[1], Oliver Kopp[2], Frank Leymann[1], and Johannes Wettinger[1]
[1]IAAS, [2]IPVS, University of Stuttgart, Stuttgart, Germany
Email:{lastname}@iaas.uni-stuttgart.de

Abstract—**In the field of cloud computing, the automated deployment of applications is of vital importance and supported by diverse management technologies. However, currently there is no systematic knowledge collection that points out commonalities, capabilities, and differences of these approaches. This paper aims at identifying common modeling principles employed by technologies to create automatically executable models that describe the deployment of applications. We discuss two fundamental approaches for modeling the automated deployment of applications: imperative procedural models and declarative models. For these two approaches, we identified (i) basic pattern primitives and (ii) documented these approaches as patterns that point out frequently occurring problems in certain contexts including proven modeling solutions. The introduced patterns foster the understanding of common application deployment concepts, are validated regarding their occurrence in established state-of-the-art technologies, and enable the transfer of that knowledge.**

Keywords–Modeling Patterns; Application Deployment and Management; Automation; Cloud Computing.

I. INTRODUCTION

Many cloud service offerings, for example, Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) offerings, and established management technologies, such as *IBM Bluemix* [1], *Chef* [2], and *Juju* [3], support the automated deployment of applications. There are also standards, for example, the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [4]. All promise high automation, reusability, and easy usage in order to operate business functionality. Contrary, using the aforementioned technologies requires that the customer describes the deployment of the application according to the languages, capabilities, and requirements of the used technology. But, all have in common that they support the same deployment automation principles that can be divided into two major modeling approaches: the *declarative* and the *imperative* deployment modeling approach.

The declarative approach uses structural models that describe the desired application structure and state, which are interpreted by a deployment engine to enforce this state. The imperative approach uses procedural models that explicitly specify a process to be executed [5][6]. These imperative process models define explicitly all activities that have to be executed, their execution order, and the data flow between these activities. Such imperative process models are executed in an automated manner by a process engine. Using the imperative approach, the customer can customize arbitrarily the deployment but typically requires considerably more expertise, for example, if multiple different cloud provider application programming interfaces (APIs) have to be invoked [5][7].

However, technologies following these approaches significantly differ in the employed domain-specific modeling languages and concepts. Whilst all technologies advertise the revolution of deploying and managing business functionality in the Cloud, up to now, there is no systematic knowledge collection that guides in choosing the right technology. As a result, for evaluating which technology fits best the customer's needs, at the moment, one has to be an expert of each considered technology for choosing the most appropriate one.

In this paper, we tackle these issues by investigating the capabilities of established deployment technologies in order to document their commonalities in the form of patterns. In particular, we investigate (i) the Cloud standard *TOSCA*, (ii) the technologies *IBM Bluemix*, (iii) *Chef*, (iv) *Juju*, and (v) *OpenTOSCA* [8], (vi) the implementations of the most downloaded artifacts in the official repositories of Chef [9] and Juju [10], and (vii) scientific publications. The chosen technologies are among the most utilized and established ones that enable application deployment in modern cloud environments that inherently require a high degree of automation. However, we do not claim that this list of analyzed technologies is complete, but nevertheless, it provides an appropriate starting point for finding new patterns that possibly occur within other approaches, standards, and technologies as well.

To overcome the problem of modeling application deployment and evaluating the best fitting technology at the same time, we first introduce pattern primitives to establish a common wording. Then, we describe the underlying deployment modeling concepts supported by the analyzed artifacts, management technologies, and scientific publications in form of the *Imperative Deployment Model* pattern and the *Declarative Deployment Model* pattern. To validate our findings, we apply Coplien's *Rule of Three* that dictates a pattern to exist in at least "*three insightfully different implementations*" [11]. Thus, we state how and where to find the pattern's implementation to prove the "Rule of Three". Using these patterns, the knowledge about application deployment principles can be transferred, e.g., , for choosing the most appropriate technology.

The remainder of this paper is structured as follows: In Section II, we define pattern primitives with which we establish a common wording for describing application deployment technologies. In Section III, we introduce the Imperative Deployment Model pattern and the Declarative Deployment Model pattern and point to their occurrences. In Section IV, we discuss the background of the paper and the related work of the pattern community and cloud computing community. In Section V, we validate our patterns. Finally, in Section VI, we conclude the results of this paper and outline future work.

II. Pattern Primitives

In this section, we define *pattern primitives* that are identified as atomic parts in the domain of application deployment. Similar to Zdun et al. [12] and Fehling et al. [13], we use the concept of pattern primitives to describe certain elements inside patterns that have specific names and characteristics. These elements are known to domain experts and may exist in other domains under different names. Thus, this section aims to establish a common wording for a precise communication and to describe the patterns we introduce in the next section.

Application: An *application* comprises software that implements a certain business functionality. Applications typically consist of multiple *software components* that are working together to realize the desired functionality. The interplay of the components may be locally or realized via network, i.e., the collaboration of components can be arbitrarily complex.

Software component: A *software component* is a part of an application that may be reused within the same application, other applications, or other companies. Components can be divided into either *application-specific software components* and *general-purpose software components*, see next.

Application-specific software component: An *application-specific software component* is a piece of software that implements a certain piece of the business functionality of an application. Such a component is highly adapted for and integrated into a certain application and implements specific functionality. Thus, application-specific components often cannot be reused within other applications due to their specialization. One example for such components are customized enterprise resource planning software components.

General-purpose software component: A *general-purpose software component* is a piece of software that implements a functionality that can be reused by many different applications for general purposes. Thus, they are explicitly made for reuse and provide common functionality that is independent of a certain business logic. Examples for such components are web servers or database management systems.

Application environment: The term *application environment* comprises all running software and hardware components of one concrete deployment of the application on all layers, i.e., physical servers, virtual machines running on these servers, operating systems, installed web servers, etc. Thus, if a certain application is deployed multiple times in different clouds, each of these deployments forms one application environment.

Management environment: In contrast to the application environment, in which an application is running in, the term *management environment* comprises all physical components, such as servers and software components, that are employed for running *deployment & management systems*, see next.

Deployment & management system: A *deployment & management system* provides the functionality for deploying, operating, and managing applications in an automated fashion, e.g., to install, configure, or terminate applications or an application's components. Deployment & management systems are running in management environments and, therefore, are typically running and operated independently from applications. There are many different flavors of deployment & management systems: Some interpret declarative models that define the structure of the desired application deployment, others are based on imperative process models that define each step that has to be executed to realize a certain deployment task, e.g., to install the entire application. We detail these two flavors in the following sections in the form of the patterns we present.

Deployment logic: The *deployment logic* describes all steps that have to be executed to deploy all components of an application. To implement the deployment logic, different levels of abstractions can be differentiated depending on the chosen form of implementation. For example, a workflow may be created that specifies a set of *deployment tasks* and their execution order while *deployment operations* implement these deployment tasks. We detail this in the following primitives.

Deployment task: A *deployment task* denotes the task of deploying a certain software component, for example, installing and configuring an Apache web server on a running Ubuntu virtual machine. To implement a deployment task in a way that enables its automated execution, typically multiple *deployment operations* have to be executed, see next.

Deployment operation: A *deployment operation* is an automatically executable piece of software that implements a certain deployment functionality, for example, to install a software package on an operating system or to configure the HTTP-port. Thus, typically multiple deployment operations are required to execute a deployment task. Deployment operations can be implemented using various kinds of technologies, for example, in the form of scripts that are executed in the application environment to install a web server on a running virtual machine or as Java programs that are executed in the management environment to orchestrate a set of API calls.

III. Patterns for Modeling the Automated Deployment of Applications

In this section, we introduce the *Imperative Deployment Model* pattern and the *Declarative Deployment Model* pattern that describe two different flavors for modeling the deployment of applications. The main purpose of applying these *modeling patterns* is to create models that can be executed automatically to deploy a certain application. Thus, the introduced patterns help to avoid manual steps executed by humans, which is mandatorily required in the domain of cloud computing, where rapid application deployment is of vital importance.

The patterns are structured to comprise information that are derived from best practices in the pattern community [11][14]-[19]: Each pattern has a *name* and a catchy *icon* to foster memorability. The *problem* statement defines the obstacle to overcome. The *context* describes the circumstances under which the problem occurs. Subsequent, the *forces* describe why the problem is not trivial to solve and why basic approaches might fail. The *solution* describes the approach of how to solve the problem. The solution is accompanied by a *solution sketch* that depicts the solution. The *results* outline the outcomes of applying an implementation of the pattern. Proven occurrences of the pattern are referenced in the *know uses*. Therefore, we show that the patterns presented in this section satisfy the *Rule of Three* [11] that instructs that at least three independent implementations of the concepts described by the pattern have to be found, cf. Section VI.

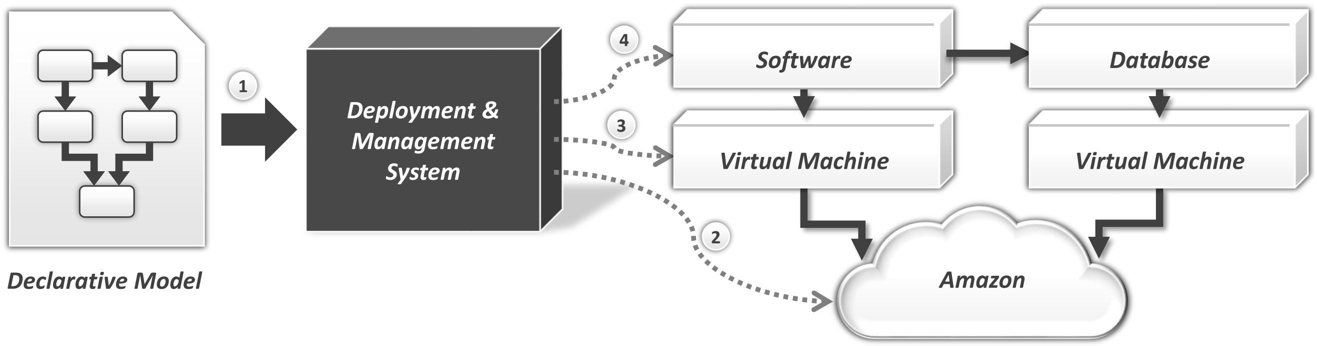
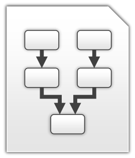


FIGURE 1. SOLUTION SKETCH FOR THE DECLARATIVE DEPLOYMENT MODEL PATTERN: A DECLARATIVE MODEL IS INTERPRETED FOR DEPLOYING A SOFTWARE IN THE AMAZON CLOUD, I.E., PROVISIONING A VM, INSTALLING SOFTWARE, AND WIRING THAT SOFTWARE WITH A RUNNING DATABASE.

A. *Declarative Deployment Model Pattern*



Problem: How to model the deployment of a simple application that requires only few or no individual customization in a way that enables its automated execution?

Context: Automate the deployment of an application.

Forces: Typically, applications comprise well-known, general-purpose software components, e.g., virtual machines running a Ubuntu, a Tomcat application server, or a MySQL server, for realizing common functionality. Such components are heavily used in industry, so they are often integrated and, as a result, it is well-known how to use them. However, a manual installation and configuration of such components is error-prone, time-consuming, and costly [7]. Thus, this is not appropriate in scenarios requiring the rapid deployment of applications and their components—especially if multiple instances of the application need to be deployed, which is a common requirement in the domain of cloud computing.

To automate this, imperative process execution technologies, such as scripts or the workflow technology [20], can be used. However, manually creating executable process models that automatically deploy the entire application is also complex and time-consuming [5]. Thus, for simple scenarios that employ common, reusable components, such as a Linux; Apache web server; MySQL database management server; or PHP, and that follow well-known application structures, spending this effort is very hard to argue and should be avoided.

Solution: Create a *declarative deployment model* that describes the structure of the application that shall be deployed, i.e., all the components as well as their dependencies and interplay. Subsequent, use a deployment & management system that understands this model and that automatically executes all required steps to deploy the application as described by the model. Declarative deployment models also specify necessary software implementations, e.g., the user interface implementation of a web application to be deployed. By modeling the deployment this way, the desired state of the application is defined, which provides the basis for the deployment & management system to automatically derive the necessary deployment tasks and operations to be executed. Thus, the system derives and executes the deployment logic automatically from the declarative model without involving the user. Systems that support this pattern are, e.g., Chef and Juju.

Figure 1 depicts the pattern's solution sketch. The declarative deployment model specifies the application structure, its components, and their interplay. The model (1) is passed to the deployment & management system that derives the required deployment logic from this model and executes all required tasks and operations. In this example, it (2) invokes the API of Amazon to create a virtual machine (VM), (3) accesses the VM to install required software packages, and (4) configures the software to connect to the installed database. Thus, the system creates the prescriptively modeled application in reality.

Results: Applying the pattern eases application deployment as no manual deployment steps are required and only a model has to be created. Moreover, the required technical skills are limited to the modeling of the declarative model. Since the pattern is primarily applicable to deployments that mainly comprise general-purpose components, which are well-known to deployment & management systems, the usage of these components is efficient and not costly as they only must be specified in the model. Moreover, by providing implementations for interfaces defined by the deployment & management system, also application-specific software components can be deployed automatically, for example, by referencing an script that installs a custom application-specific software component.

Known Uses: In Bluemix, an *App* can be described declaratively in the *manifest.yml* file containing information about the used *build pack*, amount of the App instances, and with which other *services* the App shall be bound [21]. Bluemix *boilerplates* are predefined application containers that consist of runtime environments and predefined services for a distinct purpose that can be adapted with various options, e.g., the database size [22]. Chef enables to model declaratively *cookbooks*, defining the structure by importing other cookbooks, adapting by specifying *attributes*, letting *chef-client* compile the *run-list*—the sequences of operations to execute—, and gather further requirements, e.g., other cookbooks, files, or attributes. Subsequent, the chef-client configures the virtual machine according the run-list [23]. Juju supports *bundles* describing services, their interplay, and configuration that can be provisioned without defining the distinct provisioning [24]. TOSCA enables modeling declaratively the application's structure with *Topology Templates* [4], [6]. Out of these declarative models, the imperative provisioning logic is generated [5]. The scientific deployment prototype *Engage* also enables to describe application structures for automated deployment [25].

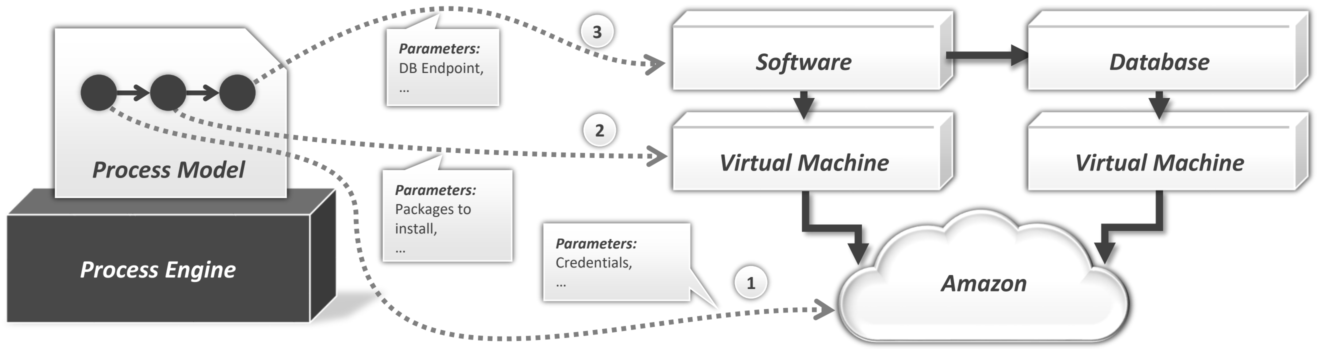
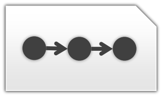


FIGURE 2. SOLUTION SKETCH FOR THE IMPERATIVE DEPLOYMENT MODEL PATTERN: A PROCESS MODEL IS EXECUTED FOR DEPLOYING A SOFTWARE IN THE AMAZON CLOUD, I.E., PROVISIONING A VM, INSTALLING SOFTWARE, AND WIRING THAT SOFTWARE WITH A RUNNING DATABASE.

B. Imperative Deployment Model Pattern



Problem: How to model the deployment of a complex application that requires application-specific customization in a way that enables its automated execution?

Context: Automate the deployment of an application.

Forces: The deployment of complex business applications that consist of various application-specific software components with complex dependencies and configurations is a serious challenge: Multiple experts have to cooperate as a significant amount of technical expertise is required and typically multiple different deployment & management systems need to be combined [7]. Thus, if an application needs to be deployed multiple times, a manual process is not possible.

Using the Declarative Deployment Model pattern is appropriate for modeling the deployment of simple applications that mainly employ general-purpose components. However, for such complex applications as described above, this pattern cannot be applied as the interpretation of declarative models in deployment & management systems cannot be influenced and customized arbitrarily [5]. Especially, when multiple deployment & management systems need to be combined, a single declarative deployment model is not possible.

Solution: Create an *imperative deployment model* that describes (i) all activities to be executed, (ii) the control flow, i.e., their execution order, and (iii) the data flow between them. Each activity implements a certain deployment task or invokes a deployment operation, e.g., an activity invokes the API of a cloud provider to provision a new virtual machine and subsequent activities copy and execute installation scripts onto this VM. Afterwards, use a *process engine* to execute the model automatically without involving the user. One robust technology for creating and executing processes is, e.g., the workflow technology [20] and standards, such as BPEL [26].

Figure 2 depicts the pattern's solution sketch. The process model is deployed on an appropriate process engine and (1) invokes the API of Amazon to create a virtual machine, (2) accesses it, e.g., via SSH, to install software packages, and (3) configures the installed software to connect to a running database that is also hosted on Amazon. Thus, if customization is required, any activity can be arbitrarily customized to invoke suitable deployment operations or other implementations.

Results: By using imperative deployment models, i.e., process models, the deployment can be modeled arbitrarily as each step to execute is specified explicitly. Thus, the model is not interpreted as in the Declarative Deployment Model pattern but executed following the model. This enables deploying general-purpose components as well as arbitrary application-specific components that require complex configurations and wirings with other components. Thus, this approach is capable of handling the complexity of arbitrary application deployments. Contrary, the deployment of such complex applications often cannot be modeled declaratively at all due to application-specific details that cannot be reflected in declarative models.

Especially the workflow technology is suited for creating complex process models as also the modeling of compensation logic is possible [27], [20]. For example, if a deployment process provisions multiple virtual machines and a failure occurs, simply stopping the process requires a manual deletion of the created VMs. By using compensation and failure handling, such cases can be explicitly considered in the process model. However, the modeling of imperative deployment logic is typically more complex for the user: With the Imperative Deployment Model pattern, a process model has to be created that explicitly defines each deployment operation to be invoked and, thus, required deep technical knowledge about the invocation and orchestration of management technologies [7]. However, this is addressed by approaches for generating such process models [5][7][28]-[31]. Moreover, there are workflow languages, such as BPMN4TOSCA [32][33], that were developed explicitly for modeling such processes.

Known Uses: For provisioning a service with Bluemix, imperative scripts can invoke depoyment tasks using the command line interface *cf* [34]. Chef-client executes the imperative *run-list* that, usually, is generated [35]. But if necessary, the run-list can be customized, e.g., by adding additional *recipes* whose actions implement deployment tasks [36]. Juju implements *hooks* that represent executable deployment tasks and are invoked in case of *events* [37]. For more direct interaction, *Actions* can be invoked with parameters to execute deployment operations [38]. TOSCA enables explicitly imperative provisioning: workflow models can be attached to services that implement the provisioning imperatively [4], [6]. The TOSCA runtime environment OpenTOSCA contains a generator for BPEL workflows that allows to generate provisioning plans that can be customized individually for certain needs [5][39].

IV. Related Work and Management Technologies

In 1979, Alexander et al. started to publish their idea to describe solutions for reoccurring problems in the domain of building architecture as patterns [15][40]. Since then, this approach has been heavily used, refined, and also been applied to the domain of IT. For example, for software developers, the principles of good object-oriented software design is captured as the patterns of the Gang of Four [16]. To foster the pattern paradigm for computer science, Buschmann et al. advanced patterns by finding patterns in the domain of IT as well as publishing their lessons learned about patterns and pattern languages [14][17]. Coplien contributed by delimiting patterns from mere copies. His *Rule of Three* states that a solution has to be implemented independently at least three times for being able to provide a base for a pattern [11].

To establish a better association between the abstract patterns and concrete pattern implementations, Falkenthal et al. introduces *Solution Implementations* that help to aggregate pattern appliances for problems that need applying multiple patterns [41][42]. Thus, these works can be used to efficiently reuse proven (declarative or imperative) deployment models as Solution Implementations of the introduced patterns.

Fehling et al. introduced patterns about how to automate certain deployment tasks in cloud computing, e.g., how to realize an elastic application [43]. Also, Fehling et al. captured reoccurring problems of migrating services to the cloud as patterns the same way [44]. The patterns' solutions are documented in the form of abstract process models that can be refined for concrete use cases. Using this approach, also proven deployment processes could be documented in an abstract manner as patterns, which are then instances of the Imperative Deployment Model pattern presented in this paper.

The methods used for findings the patterns introduced in this paper are based on the iterative approaches of how to find and author concrete patterns, introduced by Fehling et al. [13] and Reiners [18]. Wellhausen et al. introduced a concrete pattern structure, described in detail the interrelation of the pattern structure's distinct sections, and provided a step-by-step guide for improving the formulation of patterns and helping first-time authors to concisely express their patterns [19].

In 2013, the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) was published in version 1.0 [4]. TOSCA explicitly supports both deployment flavors by allowing modeling application topologies and specifying workflow models for deployment. Thus, the academic open-source prototype OpenTOSCA implements the TOSCA standard and, therefore, supports both patterns [39][8]. Since years, there are established technologies as well as recently emerging ones that help putting business functionality into the cloud. For example, there are IBM Bluemix [1], Chef [2], and Juju [3] that all support declarative as well as imperative mechanisms at different points in application deployment.

V. Validation

In this paper, we discussed pattern primitives in the domain of cloud application deployment and introduced two patterns describing fundamental principles of modeling the automated deployment of applications. In this section, we discuss the process of how we found the patterns and their validity.

Table I. Occurrence of the patterns in the technologies [45]

Occurrence	Imperative Deployment Model	Declarative Deployment Model
Bluemix	✓	✓
Chef	✓	✓
Juju	✓	✓
OpenTOSCA	✓	✓
Others	✓	✓

Usually in the pattern community, experts of a domain search for pattern candidates, discuss, and dismiss and refine them until only patterns are left. This process is very costly in time and effort. Therefore, an alternative approach distills pattern candidates and patterns from artifacts, e.g., documentation [13]. These resources can be treated as the documentation of expertise of developers and scientists. Thus, we selected a variety of application deployment approaches and technologies that are omnipresent in industry and science, their documentations, implementations of the most downloaded artifacts in their official repositories, and scientific publications as a basis for our knowledge collection. Based on the found commonalities, we elaborated the patterns iteratively according to [13]. The proposed patterns are validated regarding their occurrence. In Table I, we marked found evidences for each pattern with a ✓symbol. The row *Others* encompasses scientific publications and their prototypes. The enumeration of evidences bases partly on [45]. The concrete references to the evidences can be found in the *known uses* paragraph of the respective pattern. The *Rule of Three* states the condition of three independent occurrences of the pattern in the real world [11]. Both the Declarative Deployment Model pattern and the Imperative Deployment Model pattern fulfill this condition.

VI. Conclusion and Future Work

In this paper, we presented two modeling patterns that describe principles of modeling the deployment of applications and provide a deeper understanding of the declarative and imperative approaches. By stating details of the analyzed technologies, the patterns also foster the understanding of the analyzed standard and technologies. We proved that the documented patterns occur in many state-of-the-art deployment technologies, especially, the TOSCA standard explicitly supports both patterns. We validated the patterns by stating (i) in which artifact, documentation, standard, and technology an individual implementation of the pattern can be found and (ii) that the common pattern metric *Rule of Three* [11] is met. Thus, the defined pattern primitives and found patterns provide a means for communicating the principles in general.

The proposed Declarative Deployment Model pattern and Imperative Deployment Model pattern are the beginning of a catalog of patterns. Thus, more patterns in the domain of application deployment can be found, for example, to document proven solutions for creating imperative deployment process models. Further, the catalog can be elaborated to a full pattern language that will be addressed in our upcoming research steps. We also plan to author another kind of related patterns for the domain of application management.

Acknowledgments

This work is partially funded by the BMWi projects *NEMAR* (03ET4018B) and *SePiA.Pro* (01MD16013F).

References

[1] "IBM Bluemix – Cloud infrastructure, platform services, Watson, & more PaaS solutions," 2017, URL: https://www.ibm.com/cloud-computing/bluemix/ [accessed: 2017-02-02].

[2] "Chef – Embrace DevOps | Chef," 2017, URL: https://www.chef.io/ [accessed: 2017-02-02].

[3] "Jujucharms | Juju," 2017, URL: https://jujucharms.com/ [accessed: 2017-02-02].

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

[5] U. Breitenbücher et al., "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in International Conference on Cloud Engineering (IC2E 2014). IEEE, 2014, pp. 87–96.

[6] OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[7] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and J. Wettinger, "Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies," in On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013). Springer, 2013, pp. 130–148.

[8] "OpenTOSCA Ecosystem," 2017, URL: http://www.opentosca.org/ [accessed: 2017-02-02].

[9] "Welcome – The resource for Chef cookbooks – Chef Supermarket," 2017, URL: https://supermarket.chef.io/ [accessed: 2017-02-02].

[10] "Store | Juju," 2017, URL: https://jujucharms.com/store/ [accessed: 2017-02-02].

[11] J. O. Coplien, Software Patterns. SIGS Books & Multimedia, 1996.

[12] U. Zdun and P. Avgeriou, "A catalog of architectural primitives for modeling architectural patterns," Information and Software Technology, vol. 50, no. 9, 2008, pp. 1003–1034.

[13] C. Fehling, J. Barzen, U. Breitenbücher, and F. Leymann, "A Process for Pattern Identification, Authoring, and Application," in Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP 2014). ACM, 2014.

[14] F. Buschmann, K. Henney, and D. Schimdt, Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages. Wiley, 2007.

[15] C. Alexander, S. Ishikawa, and M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, 1994.

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley, 1996.

[18] R. Reiners, "An Evolving Pattern Library for Collaborative Project Documentation," Ph.D. dissertation, RWTH Aachen University, 2013.

[19] T. Wellhausen and A. Fiesser, "How to Write a Pattern?: A Rough Guide for First-time Pattern Authors," in Proceedings of the 16th European Conference on Pattern Languages of Programs (EuroPLoP 2011). ACM, 2012.

[20] F. Leymann and D. Roller, Production Workflow: Concepts and Techniques. Prentice Hall PTR, 2000.

[21] "Deploying apps," 2017, URL: https://www.ng.bluemix.net/docs/manageapps/depapps.html [accessed: 2017-02-02].

[22] "Boilerplates," 2017, URL: https://www.ng.bluemix.net/docs/cfapps/boilerplates.html [accessed: 2017-02-02].

[23] "About Nodes – Chef Docs," 2017, URL: https://docs.chef.io/nodes.html [accessed: 2017-02-02].

[24] "Using and Creating Bundles | Documentation | Juju," 2017, URL: https://jujucharms.com/docs/stable/charms-bundles [accessed: 2017-02-02].

[25] J. Fischer, R. Majumdar, and S. Esmaeilsabzali, "Engage: A Deployment Management System," in ACM SIGPLAN Notices. ACM, 2012, pp. 263–274.

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

[27] F. Leymann and D. Roller, "Building A Robust Workflow Management System With Persistent Queues and Stored Procedures," in Proceedings of the Fourteenth International Conference on Data Engineering (ICDE). IEEE, 1998, pp. 254–258.

[28] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Pattern-based Runtime Management of Composite Cloud Applications," in Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013). SciTePress, 2013, pp. 475–482.

[29] T. Eilam, M. Elder, A. V. Konstantinou, and E. Snible, "Pattern-based Composite Application Deployment," in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011). IEEE, 2011, pp. 217–224.

[30] K. El Maghraoui, A. Meghranjani, T. Eilam, M. Kalantar, and A. Konstantinou, "Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools," in Proceedings of the 7th International Middleware Conference (Middleware 2006). Springer, 2006, pp. 404–423.

[31] R. Mietzner, "A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing," Ph.D. dissertation, Universitt Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010.

[32] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications," in Proceedings of the 4th International Workshop on the Business Process Model and Notation. Springer, 2012, pp. 38–52.

[33] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and T. Michelbach, "A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications," in Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015. CEUR Workshop Proceedings, 2015, pp. 51–54.

[34] "CLI- und Dev-Tools," 2017, URL: https://console.ng.bluemix.net/docs/cli/index.html#cli [accessed: 2017-02-02].

[35] "About Run-lists – Chef Docs," 2017, URL: https://docs.chef.io/run_lists.html [accessed: 2017-02-02].

[36] "About Recipes – Chef Docs," 2017, URL: https://docs.chef.io/recipes.html [accessed: 2017-02-02].

[37] "Charm hooks | Documentation | Juju," 2017, URL: https://jujucharms.com/docs/stable/authors-charm-hooks [accessed: 2017-02-02].

[38] "Implementing actions in Juju charms | Documentation | Juju," 2017, URL: https://jujucharms.com/docs/stable/authors-charm-actions [accessed: 2017-02-02].

[39] T. Binz et al., "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications," in Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). Springer, 2013, pp. 692–695.

[40] C. Alexander, The Timeless Way of Building. Oxford University Press, 1979.

[41] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, "From Pattern Languages to Solution Implementations," in Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014). Xpert Publishing Services, 2014, pp. 710–726.

[42] ——, "Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains," International Journal On Advances in Software, vol. 7, no. 3&4, 2014, pp. 710–726.

[43] C. Fehling, F. Leymann, J. Rütschlin, and D. Schumm, "Pattern-Based Development and Management of Cloud Applications," Future Internet, vol. 4, no. 1, 2012, pp. 110–141.

[44] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas, "Service Migration Patterns – Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments," in Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013). IEEE, 2013, pp. 9–16.

[45] C. Endres, "A Pattern Language for Modelling the Provisioning of Applications," Master's thesis, University of Stuttgart, 2015.