Context-Aware Provisioning and Management of Cloud Applications

Uwe Breitenbücher[1(✉)], Tobias Binz[1], Oliver Kopp[2], Frank Leymann[1], and Matthias Wieland[2]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{breitenbuecher,binz,leymann}@informatik.uni-stuttgart.de
[2] Institute for Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany
{kopp,wieland}@informatik.uni-stuttgart.de

Abstract. The automation of application provisioning and management is one of the most important issues in Cloud Computing. However, the steadily increasing number of different services and software components employed in composite Cloud applications leads to a high risk of unintended side effects when different technologies work together that bring their own proprietary management APIs. Due to unknown dependencies and the increasing diversity and heterogeneity of employed technologies, even small management tasks on a single component may compromise the whole application functionality for reasons that are neither expected nor obvious to non-experts. In this paper, we tackle these issues by introducing a method that enables detecting and correcting unintended effects of provisioning and management tasks in advance by analyzing the context in which the tasks are executed. We validate the method practically and show how context-aware expert management knowledge can be applied fully automatically to provision and manage running Cloud applications.

Keywords: Application management · Provisioning · Context · Automation · Cloud computing

1 Introduction

Cloud Computing enables enterprises to outsource their IT efficiently due to properties such as pay-on-demand computing [25]. To exploit these properties for their offerings, Cloud providers have to automate their processes for application provisioning and management. Therefore, a lot of tools and management technologies have been developed. However, due to specific requirements on employed Cloud services and software components, proprietary systems of different providers often have to be combined in *Complex Composite Cloud Applications* [20]. Unfortunately, automating the provisioning and management of such

M. Helfert et al. (Eds.): CLOSER 2014, CCIS 512, pp. 151–168, 2015.
DOI: 10.1007/978-3-319-25414-2_10

applications is a difficult challenge because their management technologies typically provide proprietary and heterogeneous management APIs, security mechanisms, and data formats which need to be integrated, too [8]. This leads to a high risk of unexpected side effects when a task unintentionally affects multiple parts of an application. Thus, managing such applications requires (i) a deep technical insight in each technology and (ii) an overall understanding of the system. In many cases, only experts are able to execute management tasks correctly. However, they also reach their limits when a management task has to be executed on a complex application whose exact structure and runtime state are not documented: Unknown relations and dependencies between components that directly influence each other's functionality lead to a serious management challenge. Thus, if the *context*, in which a management task is executed, is not explicitly known, understood, and considered, there is a high risk of unintended side effects. In addition, as manually executing management task in large systems is slow, costly, and error prone, Cloud application management must be automated [14,20,30].

In this paper, we present an approach that enables applying expert management knowledge for provisioning and management tasks in a certain context automatically to running applications. We introduce an abstract (i) *Context-Aware Application Management Method* and (ii) present a fully automated realization of this method for the provisioning and management of applications to validate its practical feasibility. The method introduces *Declarative Management Description Models* (DMDM) to describe management tasks declaratively including their context in a formal model. This enables experts to detect unintended impacts and side effects of management tasks through analyzing them in the context in which they are executed. We show that an individual context analysis is often required due to the heterogeneous nature of the involved components and management technologies—which is not possible using imperative approaches such as workflows or scripts. The automated realization of the method validates the method's practical feasibility. It enables organizations to operate a variety of different applications consisting of heterogeneous components without the need to employ or educate specialized experts that have the required technical knowledge. This paper is an extended version of a former work [10] we presented at the *4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. While the former paper considers only application management, we show in this paper how the method can be used also for the provisioning of applications by introducing *Automated Provisioning Patterns*.

In the next section, we describe limitations of existing management automation approaches and present a motivating scenario in Sect. 3. Section 4 presents the method, which is automated in Sect. 5. In Sect. 6, we present the paper's new contribution: We apply the method to application provisioning and introduce Automated Provisioning Patterns. In Sect. 7, we describe related work. Section 8 concludes the paper and gives an outlook on future work.

2 Limitations of Imperative Management Approaches

To automate application management, the execution of management tasks is often described imperatively using executable processes implemented as workflows [26], scripts, or programs. If an application is crucial to the business of an enterprise, errors that possibly result in system downtime are not acceptable. Therefore, often only the robust and reliable workflow technology can be used that provides features such as fault handling and compensation mechanisms [26]. Nevertheless, before executing such workflows, they must be verified to ensure a correct implementation. Unfortunately, the context, in which the management tasks are executed, is not explicitly described and, thus, not visible in such processes. As a result, management processes cannot be analyzed by experts in consideration of the management tasks' context as only operation calls, service invocations, or script executions on the *directly* affected components are described in workflows, but not the surrounding environment: Experts see only the directly affected part of the application, not the whole application structure. Thus, other application components that may be affected indirectly, too, cannot be considered in this analysis. For example, if the database of a Web-based application shall be replaced by a database from a different vendor, the application's Web Server may require a certain database connector to be installed for connecting to the new database. If this dependency is not considered and handled by the management workflow that replaces the database through installing the required connector, too, the application cannot connect to its database anymore. This quickly results in system downtimes caused by errors that are neither easy to find nor to fix. Thus, the most important requirement to enable context-aware management is a formal model that describes both the management tasks as well as their context.

3 Motivating Scenario

In this section, we describe the motivating scenario that is used to explain the proposed method and its realization. The scenario describes a business application that consists of a PHP-based Web frontend and a PostgreSQL database. The frontend shall be migrated from one Cloud to another. Because the application evolved over time, it is currently hosted on two Clouds: The PHP frontend is hosted on Microsoft's public Cloud offering "Windows Azure", the PostgreSQL database on Amazon's PaaS offering "Relational Database Service (RDS)". The PHP frontend runs on an Apache HTTP Server (including PHP-module) which is installed on an Ubuntu Linux operating system that runs in a virtual machine hosted on Azure. The management task that has to be executed is migrating the PHP frontend to Amazon's IaaS offering "Elastic Compute Cloud (EC2)" to reduce the number of employed Cloud providers. This migration results in two issues that compromise the application's functionality if they are not considered in advance: (i) Missing database driver and (ii) missing configuration of

the database service. To migrate the PHP frontend, we have to create a new virtual machine on Amazon EC2, install the Apache HTTP Server and the PHP-module, and deploy the corresponding PHP files. This works without further configuration issues. However, connecting the PHP application to the database is not as easy as it seems to be: Simply defining the database configuration of the PHP frontend by setting the database's endpoint, username, and password is not sufficient. Here, a technical detail of the underlying infrastructure needs to be considered: The PHP-module of the Apache HTTP Server needs different database drivers to connect to different types of databases. Thus, if the PostgreSQL driver gets not installed explicitly on the server, the PHP frontend is not able to connect to the database. However, this is not easy to recognize as applications often employ MySQL databases whose drivers are typically installed together with the PHP-module. Thus, installing the required driver for PostgreSQL might be forgotten. The second issue is even more difficult to foresee if the administrator is not an expert in Amazon RDS: Databases running on Amazon RDS are per default not accessible from external components. To allow connections, a so-called "Security Group" must be defined to configure the firewall. This group specifies the IP-addresses which are allowed to connect to the database. Both issues result in breaking the application's functionality as the frontend can not connect to the database. The reason for both problems lies in ignoring the context in which the tasks are executed: (i) If an application shall connect to a certain database, the application's runtime environment must support this kind of database. (ii) Accessing a database hosted on Amazon RDS requires also more than simply writing endpoint information into a configuration file as the firewall of the service has to be configured, too. Thus, for these tasks, the context in the form of the infrastructure that hosts the database and the database type has to be considered to recognize the problems. However, both problems cannot be detected if the migration is implemented using traditional approaches such as management workflows or scripts: A wrong process possibly models only the steps for (i) shutting down the old virtual machine on Azure, (ii) creating the new virtual machine on Amazon EC2, (iii) installing the Apache Web Server and the PHP-module, (iv) deploying the frontend, and (v) setting the database's IP-address, name, port, username, and password in the frontend's configuration. However, this process neither provides information about the database's type nor which infrastructure is employed. Thus, the context, in which the management tasks are executed, is not described and the problems can not be detected.

4 Context-Aware Application Management Method

The *Context-Aware Application Management Method* provides a means to consider the context in which management tasks on application components or relations are executed. The method is shown in Fig. 1 and separates between a declarative description of the management tasks to be executed and the final executable management process. In the following subsections, we explain each step in detail.

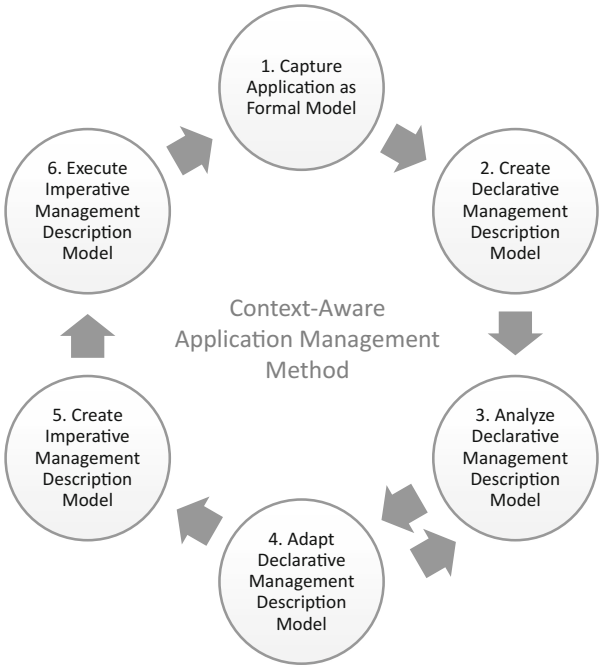

Fig. 1. Context-Aware application management method.

4.1 Step 1: Capture Application as Formal Model

First, the application to be managed is described as a formal model. This model captures the application structure and its state, i.e., (i) all components such as Web Servers, virtual machines, or installed applications, (ii) the relations between them, e.g., database connections, and (iii) their runtime information. The semantics of these *model elements* are described using types, e.g., a component may be of type "ApacheHTTPServer", a relationship of type "SQLConnection". To enable a precise definition of the elements, types can be inherited: The "ApacheHTTPServer" type is a subtype of "HTTPServer". Runtime information is described as element properties, e.g., the "ApacheHTTPServer" has the properties "IP-Address" and "Port" that specify its endpoint. Their schema is defined by the type of the element. This formalization of the running application provides a detailed, structured, and machine readable means to document a current snapshot of the application structure and all runtime information.

4.2 Step 2: Create Declarative Management Description Model

In the second step, the desired management tasks are described based on the formal model. Therefore, we introduce the *Declarative Management Description Model* (DMDM) that extends the formal model captured in Step 1 by a declarative description of the management tasks to be executed on components and

relations. This model declares management tasks in an abstract manner without technical implementation details and specifies the target component or relation of each task. A DMDM is not executable as it describes only *what* has to be done, but not *how*—all technical details are missing. For example, a DMDM may declare a "Create" task on an added relation of type "SQLConnection" between a PHP application and a SQL database, which means that the connection has to be established. However, it provides neither technical implementation details nor specifies the control flow between multiple different management tasks.

4.3 Step 3: Analyze Declarative Management Description Model

The DMDM created in the previous step captures a snapshot of the application and the abstract management tasks to be executed. The model describes the whole *context* in which tasks are executed by modelling all components and relations of the application that might be affected. In the third step, management tasks are analyzed in their context by experts of different domains to detect unexpected impacts leading to unintended side effects. DMDMs enable cooperation between different experts and separate concerns based on a uniform, structured, and formal model: Apache HTTP Server experts are able to detect that the installation of a certain database connector is required, experts of the Amazon Cloud are able to configure the Security Group in order to allow connections from the external PHP frontend of the application. Thus, DMDMs can be analyzed by multiple experts of different domains in a cooperative manner.

4.4 Step 4: Adapt Declarative Management Description Model

After the expert analysis, found problems have to be resolved to achieve the desired management goals. Therefore, the DMDM is adapted in this step by the respective experts to enable a correct execution of the tasks: Components, relations, and tasks of the DMDM may be added or reconfigured. For example, the missing database connector found in the analysis of the previous step is resolved by adding the task to install the required connector on the Web Server. Thus, each task was verified in its respective context in the previous step and gets corrected if necessary in this step. However, if tasks are added or reconfigured, all tasks have to be analyzed again for correctness as the context changes through this adaptation. This may lead to new problems and unintended side effects on other components or relations that have to be found. Therefore, Step 3 and Step 4 are repeated until no new problems are found and all tasks were considered in the final context. This ensures that also the adaptations are checked.

4.5 Step 5: Create Imperative Management Description Model

The verified and adapted Declarative Management Description Model resulting from the previous step describes the tasks to be performed declaratively in an abstract manner—only *what* management tasks have to be performed, but

not *how*. Thus, the model is not executable as the technical realization is not described. Therefore, an executable process model that implements the management tasks declared in the DMDM must be created. As this process model *imperatively* describes how the tasks have to be executed, we call these management processes *Imperative Management Description Models* (IMDM). An IMDM can be executed using an appropriate process engine and describes also the control flow and data handling between the management tasks. The IMDM has to implement exactly the semantics of the management tasks described by the adapted Declarative Management Description Model resulting from the previous step.

4.6 Step 6: Execute Imperative Management Description Model

In the last step, the IMDM is executed to perform the desired management tasks on the real running application. Therefore, a process engine is employed to run the process. As a result, the changes described by the tasks are applied to the running application in consideration of the context.

5 Realization and Validation

The presented method enables combining *declarative* management descriptions, which include all relevant context information to verify the tasks, and *imperative* processes, which are employed to actually perform the tasks on running applications. Thus, it combines two different types of Management Description Models which enables benefiting from advantages of both worlds. Therefore, the presented method provides the basis for enabling automated context-aware application provisioning and management. In this section, we validate the proposed method by showing a fully automated implementation using existing frameworks. We describe our prototypical realization for all steps of the method in the following.

5.1 Formalizing Applications Using Enterprise Topology Graphs

In Step 1, the application structure and runtime information have to be captured as formal model. We use *Enterprise Topology Graphs* (ETG) [5] as model language as they are a common way to formalize such information. ETGs are directed graphs that describe the application's structure as topology model that contains each component as typed node and each relation as typed edge. Runtime information is captured as properties of the respective model element. Thus, ETGs can be used to model the context in which a management task is executed. As ETGs support the XML-format, they are machine readable. On the left of Fig. 2, the ETG of the motivating scenario is shown. Binz et al. showed that ETGs of running applications can be discovered fully automatically using the ETGs *Discovery Framework* [3]. Thus, the first step of formalizing the application to be managed can be automated by using this framework.

5.2 Automating the DMDM Creation

Capturing application snapshots as ETG models provides a means to describe the context in which a management task is executed. Therefore, to create the DMDM in Step 2, we use the discovered ETG and annotate the management tasks to be executed directly at the affected components and relations of the ETG. In Breitenbücher et al. [6], we introduced so-called *Desired Application State Models*, which provide exactly this type of model for describing tasks to be executed declaratively in the context in which they have to be executed based on ETGs. Figure 2 shows the Desired Application State Model that describes our migration motivating scenario (rendered using *Vino4TOSCA* [11]). The colored circles with the symbols inside represent the management tasks to be executed in the form of so-called *Management Annotations* [6]. A Management Annotation describes a task to be performed in a declarative way: It defines only the type of the task and possible configuration properties, but not how to execute it. The green colored "Create-Annotations" with the star inside declare that the corresponding elements have to be created, whereas the red colored "Destroy-Annotations" with the "x" inside declare that the elements have to be destroyed. Management Annotations can be also bound declaratively to non-functional requirements in the form of policies that must be fulfilled when executing the task [9,13]. Annotating management tasks to ETGs, i.e., creating a DMDM, can be automated, too: Desired Application State Models can be generated by applying so-called *Automated Management Patterns* to ETGs [6]. An Automated Management Pattern consists of a (i) Topology Fragment and a

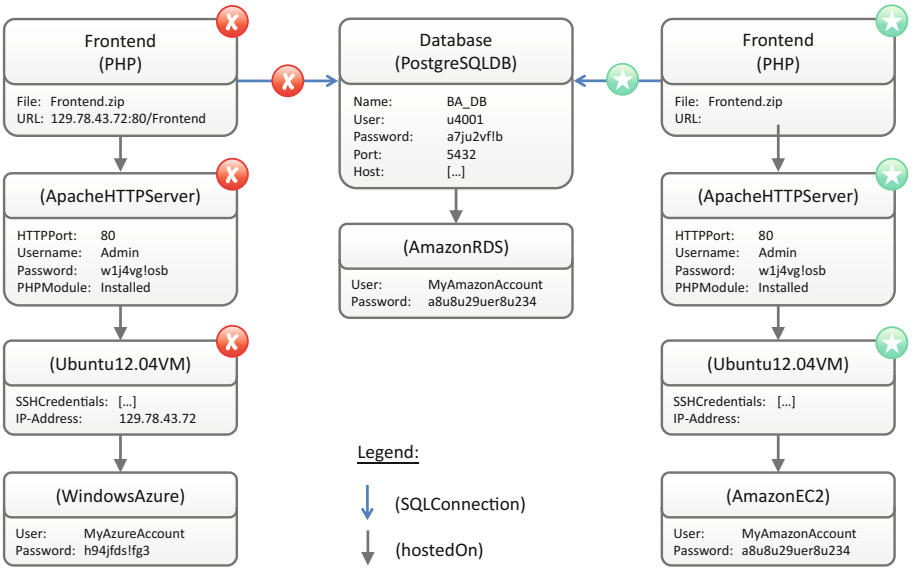

Fig. 2. Desired Application State Model after applying the idiom (simplified).

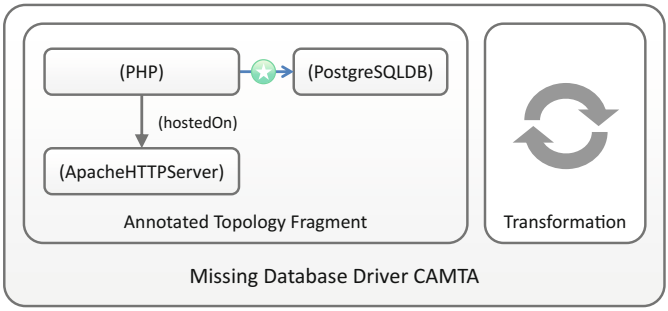

Fig. 3. CAMTA that recognizes the problem of missing PostgreSQL database connector.

(ii) Topology Transformation. The *Topology Fragment* describes the application structure to which the pattern can be applied. Thus, the pattern can be applied to all ETGs that match this fragment. The *Topology Transformation* implements the pattern's solution as executable transformation that automatically annotates the Management Annotations to be executed to the input ETG. We distinguish between *Semi-Automated Management Patterns*, which provide an abstract solution of a pattern that must be refined for concrete use cases manually, and *Fully-Automated Management Idioms*, which provide an already refined solution [7]. For example, the Desired Application State Model shown in Fig. 2 is the result of applying the "Migrate PHP Application to Amazon EC2 Idiom". Thus, the only manual step is selecting a Fully-Automated Management Idiom.

5.3 Context-Aware Task Analyzer

After the Desired Application State Model was created automatically by applying an Automated Management Pattern, it has to be analyzed by experts in Step 3 and adapted if necessary in Step 4. As we aim for automating the whole method realization, also these two steps need to be automated. Therefore, we introduce the concept of *Context-Aware Management Task Analyzers* (CAMTA) that provides a means to capture context-aware expert management knowledge in a form that enables a fully automated application to the Desired Application State Model resulting out of the previous step. The notion of CAMTAs is detecting and correcting problems by analyzing the tasks in their context and adapting the model if necessary fully automatically without manual interaction. Therefore, a CAMTA consists of two parts: (i) An *Annotated Topology Fragment* and a (ii) *Transformation*, similarly to Automated Management Patterns. The Annotated Topology Fragment is a small topology that specifies the management tasks in a certain context for which the CAMTA is able (i) to analyze correctness and (ii) to provide expert management knowledge required to adapt the model if necessary. The fragment is used for matchmaking of CAMTAs and Desired Application State Models: If the CAMTA's fragment matches elements

and Management Annotations in the model, the Context-Aware Management Task Analyzer is able to analyze exactly that part. Thus, the Annotated Topology Fragment is used to select the CAMTAs that have to be applied to analyze the DMDM in Step 3 fully automatically. For adapting the model in Step 4, each CAMTA implements a context-aware transformation that transforms the input Desired Application State Model fully automatically if necessary. Therefore, the transformation checks if the tasks specified in the CAMTA's Topology Fragment can be executed safely: If yes, the transformation returns the unmodified model. If not, the transformation adds or configures components, relationships, or tasks for correcting the Desired Application State Model. Figure 3 shows a CAMTA that analyzes the tasks of establishing a SQL connection from a PHP application hosted on an Apache HTTP Server to a PostgreSQL database. The shown CAMTA is able to analyze if establishing a SQLConnection in the context of a PHP Application running on the Apache HTTP Server to a PostgreSQL database is possible. This is expressed by its Annotated Topology Fragment on the left. The transformation shown on the right analyzes the Desired Application State Model, finds out whether the PostgreSQL connector driver is missing, and adds the corresponding model elements and tasks to the model if necessary. Thus, based on two CAMTAs, the Desired Application State Model, which results from applying a Fully-Automated Management Idiom, gets adapted fully automatically to resolve the issues of the missing database connector and Security Group configuration. The respective CAMTAs insert two different Management Annotations into the Desired Application State Model: (i) A "ConfigureSecurityGroup-Annotation" that is attached to the Amazon-RDS node and an "InstallDriver-Annotation" attached to the Apache HTTP Server node. The ConfigureSecurityGroup-Annotation configures the Amazon-RDS instance in a way that the database is accessible by the Apache HTTP Server. The InstallDriver-Annotation declares that the required connector for PostgreSQL databases must be installed. As Desired Application State Models typically specify multiple tasks to be executed in the form of Management Annotations that need to be analyzed in their context, multiple different CAMTAs are needed to check the correctness of the whole model. As they may change the model, all CAMTAs need to be applied every time after one CAMTA transformed the model to ensure that all Management Annotations are validated in the current context. As soon as input and output model do not change anymore after applying all matching CAMTAs, Step 4 is finished.

5.4 Management Plan Generation

After the DMDM was analyzed for correctness and adapted in the previous steps, the resulting model is not executable as it describes the tasks to be performed only declaratively, i.e., without implementation and control flow: The DMDM has to be transformed into an executable imperative model in Step 5. Therefore, we employ the Management Planlet Framework presented in Breitenbücher et al. [6] that employs *Management Planlets* to translate Desired Application State Models fully automatically into executable BPEL workflows. Management Planlets

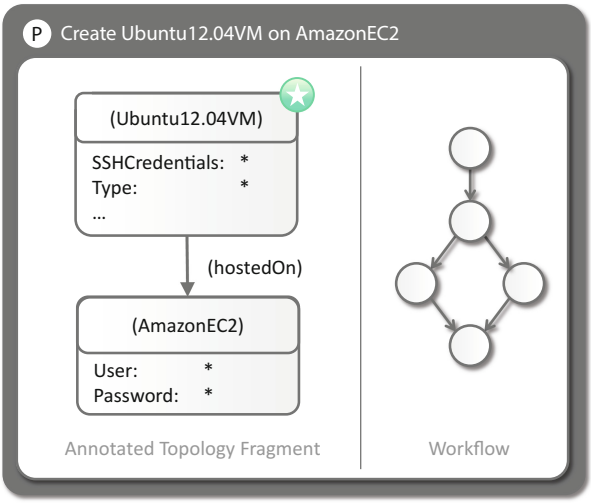

Fig. 4. Management Planlet that creates an Ubuntu12.04 virtual machine on Amazon's infrastructure service Elastic Compute Cloud (EC2).

provide the low-level imperative management logic to execute the declarative Management Annotations used in Desired Application State Models and support defining functional as well as non-functional requirements [9,13]. They serve as generic management building blocks that can be orchestrated to implement a higher-level management task. A Management Planlet consists of two parts: (i) *Annotated Topology Fragment* and (ii) a *workflow*. The fragment exposes the Planlet's functionality and is used to find Planlets that are capable of executing the specified management tasks in the respective context. For example, the Planlet shown in Fig. 4 is capable of executing the Create-Annotation attached to an Ubuntu12.04VM node if this node has to be hosted on AmazonEC2. The Planlet's workflow implements exactly the management logic required to create this virtual machine on EC2. Based on these fragments, Planlets can be orchestrated to an overall management plan that performs all annotations defined in the Desired Application State Model. Therefore, the framework employs a Plan Generator that transforms Desired Application State Models into executable workflows.

6 Context-Aware Cloud Application Provisioning

In this section, we present the new contribution of this paper that focuses on the context-aware provisioning of applications. We show how the Context-Aware Application Management Method presented in Sect. 4 can be used also for context-aware provisioning and show afterwards how this variant of the method can be automated by introducing the concept of *Automated Provisioning Patterns*.

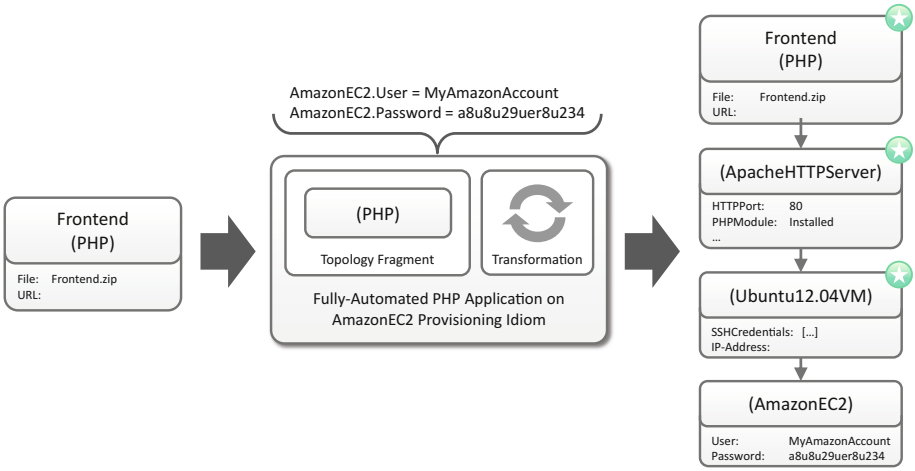

Fig. 5. Transformation of an application topology model (left) to a Desired Application State Model (right) by applying a Fully-Automated Provisioning Idiom.

6.1 Provisioning Variant of the Method

The original method for management consists of six steps and starts with capturing the application to be managed as formal model describing the application's structure and state. This model provides the entry point to define the management tasks to be executed. In our realization, all following steps are based on the original ETG that provides the basic context. In terms of provisioning, such an instance model does not exist as no ETG is available for non-provisioned applications. Therefore, the first step has to be removed if the method shall be applied to application provisioning. Thus, the method directly starts with creating the DMDM. As a result, we define Step 1 of the method as optional to enable using the method for both application provisioning as well as executing management tasks—the following steps are identical. However, to automatically create the DMDM for the provisioning in the form of a Desired Application State Model, this requires a special kind of pattern since Automated Management Patterns and Idioms require an input ETG to be transformed. Therefore, we introduce Automated Provisioning Patterns in the following subsection.

6.2 Automated Provisioning Patterns and Provisioning Idioms

We distinguish also for the provisioning between patterns and idioms by introducing (i) Semi-Automated Provisioning Patterns and (ii) Fully-Automated Provisioning Idioms. Both consist of a (i) Topology Fragment and a (ii) Topology Transformation, similar to the management approach. However, their input is not the ETG of a running application but an *application topology model* of the desired application. This model is either empty or describes single nodes and relations of the desired application deployment—but without runtime information.

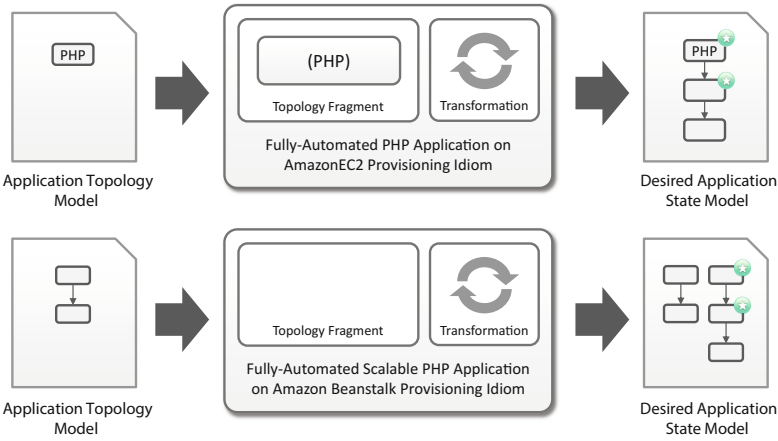

Fig. 6. Two classes of Automated Provisioning Patterns/Idioms: topology-dependent (top) and topology-independent (bottom).

The Topology Fragment is matched against the topology model and the transformation works on the matching elements. *Semi-Automated Provisioning Patterns* generate Desired Application State Models that need to be refined afterwards, i.e., they insert nodes or relations of abstract types that must be refined to a concrete type manually. In addition, added Management Annotations may need to be configured or additional annotations may need to be added. For example, an inserted abstract "InfrastructureService" node must be refined manually to a concrete node type, e.g., "AmazonEC2". *Fully-Automated Provisioning Idioms* generate already refined Desired Application State Models that can be used directly for the plan generation. For example, Fig. 5 shows an idiom for hosting a PHP application on EC2. The idiom consumes the application topology, which was created manually by an administrator in Step 2, and requests user and password of the Amazon account as input. The topology model contains only a PHP node describing the files to be deployed. The idiom's transformation inserts concrete infrastructure nodes, relations, and Management Annotations that are already refined for this concrete use case. Thus, the resulting Desired Application State Model can be used directly for the plan generation. In Step 2, multiple provisioning patterns and idioms can be applied to build complex applications.

6.3 Topology-Dependent and Topology-Independent Patterns

In this section, we present two different classes of Automated Provisioning Patterns: (i) Topology-dependent and (ii) topology-independent. We do not distinguish between patterns and idioms in this section because this difference is not important for the following considerations. Therefore, we refer to both as Automated Provisioning Patterns. The first class of *topology-dependent* patterns specify a Topology Fragment that must match corresponding elements in an

application topology model to which the pattern shall be applied. For example, the idiom shown in Fig. 6 on the top is in this class and can be applied to all application topology models that contain PHP nodes. This kind of Automated Provisioning Patterns can be used to complete or change an incomplete application topology model including the specification of the corresponding Management Annotations to be executed to provision the model. Thus, they might really *transform* an application topology into a Desired Application State Model, i.e., they may change properties of already specified nodes and relations, add or remove nodes and relations, and insert the required Management Annotations. In contrast to this, *topology-independent* patterns do not specify a Topology Fragment. Thus, they can be applied to every application topology model, even to empty ones that do not specify any node or relation at all. Patterns in this class only *insert* new nodes, relations, and annotations to create a Desired Application State Model but do not change the existing elements, as shown in Fig. 6 on the bottom. This kind of Automated Provisioning Patterns can be used to capture complete application architecture templates that can be inserted at once without transforming the original topology model elements. For example, a complete scalable LAMP (Linux, Apache, MySQL, PHP) stack hosted on a certain Cloud provider can be implemented as Fully-Automated Provisioning Idiom.

7 Related Work

Context-aware systems adapt their functionality and behaviour using context information about the environment. An often used definition for context was given by Dey [15]: "Context is any information that can be used to characterize the situation of an entity, where an entity can be a person, place, physical or computational object". An important type of context information, which is often neglected, is the state and structure of an application to be managed. In this paper, we use this type of context information to verify, configure, and execute management tasks on applications and their infrastructure. The automated realization of the presented management method provides, therefore, the basis to implement Context-aware Cloud Application Management Systems.

To model and manage context information, many frameworks have been developed in the past years. There are simple, widget-like frameworks for sensor information such as the Context Toolkit [16] and systems that support smart environments like Aura [23] or Gaia [31]. Different types of development frameworks, e.g., the framework of Henricksen and Indulska [22], and context management platforms, e.g., the Nexus Platform [21], were developed that aim at efficient provisioning of context information within a global scope. These frameworks use *Context Models* as an abstraction layer between applications and the technical infrastructure that gathers the context data. However, there is no framework that manages context information for application management in the form of the Declarative Management Description Models introduced in this paper, which provide a kind of Context Model that (i) enables capturing the environment in which management tasks are executed and (ii) the management tasks themselves described in a declarative fashion. In the realization, the

context is captured in a domain-specific data structure in the form of ETGs. Furthermore, no sensors integration has to be achieved because the context is detected on the fly using the ETG Discovery Framework [3]. Thus, the context is always up to date and does not have to be stored or managed using additional tooling.

There are several approaches that enable describing application topologies including runtime information and dependencies. Scheibenberger and Pansa [32] present a generic meta model to describe resource dependencies among IT infrastructure components. They separate the static view, which captures functional and structural aspects, from the dynamic operational view, which captures runtime information. In contrast to the employed concept of ETGs in the validation, their approach enables to model dependencies between component properties. The method's realization may be extended to capture also such fine-grained dependencies if necessary that may help experts to analyze possible impacts of a certain management task. The *Common Information Model* (CIM) [17] is a standard that provides an extensible, object-oriented data model used to capture information about different parts of an enterprise. It also provides a specification to describe application structures including dependencies. However, all these works may be used to formalize the application structure, dependencies, and runtime information, but they provide no means to model also the management tasks to be executed as required to implement a DMDM.

There are several frameworks that employ declarative descriptions to generate workflows such as Eilam et al. [18], Maghraoui et al. [28], and Keller et al. [24]. The first two focus mainly on provisioning of applications whereas the third also considers application management. In general, the proposed method can be adapted and applied to all approaches that transform declarative descriptions into imperative processes. However, it must be ensured that the declarative descriptions (i) provide the whole context and (ii) that the management tasks to be executed are described by this model somehow. In a former work [12], we showed how declarative provisioning descriptions can be transformed automatically into imperative workflows based on the TOSCA standard [4,29]. The application to be provisioned is described as topology model describing all application components and relations. As the tasks to be executed are obvious and the whole context of the provisioning is provided by this model in the form of the topology, the method can be adapted for this standards-based provisioning approach, too.

There are several pattern-based approaches that focus on the automation of application provisioning and deployment. For example, Lu et al. [27] use patterns to automate the deployment of applications. However, they employ *model-based patterns* that are different from the kind of patterns and idioms we consider in this paper. Their patterns are defined as topology models that are used to associate or derive the corresponding logic required to deploy the combination of nodes and relations described by the topology, similarly to our concept of Management Planlets. Fehling et al. [19] show how architectural Cloud patterns can be applied using a provisioning tool. However, all available approaches do

not generate models that declaratively specify the abstract management tasks to be executed following a concept such as Management Annotations. Nevertheless, as the context is typically provided by the employed models, the general idea of the method can be applied to most of these approaches, too.

The model-driven SOA deployment platform presented by Arnold et al. [1,2] supports formally capturing topology-based deployment models at different levels of abstraction—ranging from abstract models, which they call *patterns*, to *concrete* models. This classification is similar to our approach of differentiating patterns and idioms and enables non-expert administrators to safely compose and iteratively refine deployment patterns, which results in fully-specified topologies with bindings to concrete resources. However, in contrast to our automated patterns and idioms, their patterns and concrete models capture only the structure and constraints of a composite solution and do not specify the management or provisioning tasks to be executed. In Arnold et al. [2], they present an approach how these patterns can be realized automatically and introduce *Parameterized Reconfiguration Patterns* that are conceptually similar to our Automated Provisioning Patterns: They define preconditions in the form of existing model elements and specify new elements to be provisioned. Similarly, Parameterized Reconfiguration Patterns also define input parameters that are used to configure the provisioning. The result of applying such patterns are models specifying the desired application state, but without the tasks to be executed. Nevertheless, the general idea of the method can be applied to this approach, too.

8 Conclusions

In this paper, we introduced an abstract Context-Aware Application Management Method that enables applying context-aware provisioning and management expertise. We showed that separating models for context-aware analysis and management task execution provides a powerful means to benefit from advantages of both worlds. Therefore, we employed abstract Declarative Management Description Models for describing the context as well as the management tasks to be executed themselves that are transformed into Imperative Management Description Models. The presented method is validated by an automated prototypical realization for application provisioning and management using the Management Planlet Framework. We plan to integrate non-functional requirements into the method and its realization and to apply both to the OASIS standard TOSCA.

Acknowledgements. This work was partially funded by the BMWi project CloudCycle (01MD11023).

References

1. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Pattern based SOA deployment. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 1–12. Springer, Heidelberg (2007)

2. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Automatic realization of SOA deployment patterns in distributed environments. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 162–179. Springer, Heidelberg (2008)
3. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: Automated discovery and maintenance of enterprise topology graphs. In: SOCA 2013, pp. 126–134. IEEE, December 2013
4. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: Bouguettaya, A., Sheng, Q.Z., Daniel, F. (eds.) Advanced Web Services, pp. 527–549. Springer, New York (2014)
5. Binz, T., Fehling, C., Leymann, F., Nowak, A., Schumm, D.: Formalizing the cloud through enterprise topology graphs. In: CLOUD 2012, pp. 742–749. IEEE, June 2012
6. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Pattern-based runtime management of composite cloud applications. In: CLOSER 2013, pp. 475–482. SciTePress, May 2013
7. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Automating cloud application management using management idioms. In: PATTERNS 2014, pp. 60–69. IARIA Xpert Publishing Services, May 2014
8. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated cloud application provisioning: interconnecting service-centric and script-centric management technologies. In: Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., De Leenheer, P., Dou, D., Meersman, R. (eds.) ODBASE 2013. LNCS, vol. 8185, pp. 130–148. Springer, Heidelberg (2013)
9. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wieland, M.: Policy-aware provisioning of cloud applications. In: SECURWARE 2013, pp. 86–95. IARIA Xpert Publishing Services, August 2013
10. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wieland, M.: Context-aware cloud application management. In: CLOSER 2014, pp. 499–509. SciTePress, April 2014
11. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: a visual notation for application topologies based on TOSCA. In: Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., Meersman, R., Panetto, H. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 416–424. Springer, Heidelberg (2012)
12. Breitenbücher, U., et al.: Combining declarative and imperative cloud application provisioning based on TOSCA. In: IC2E 2014, pp. 87–96. IEEE, March 2014
13. Breitenbücher, U., et al.: Policy-aware provisioning and management of cloud applications. Int. J. Adv. Secur. **7**(1&2), 15–36 (2014)
14. Brown, A.B., Patterson, D.A.: To err is human. In: EASY 2001, p. 5, July 2001
15. Dey, A.K., Abowd, G.D., Salber, D.: Managing Interactions in Smart Environments. A Context-Based Infrastructure for Smart Environments, pp. 114–128. Springer, London (2000)
16. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum. Comput. Interact. **16**, 97–166 (2001)
17. Distributed Management Task Force: Common Information Model (2010)
18. Eilam, T., et al.: Pattern-based composite application deployment. In: Integrated Network Management, pp. 217–224. IEEE (2011)

19. Fehling, C., Leymann, F., Retter, R., Schumm, D., Schupeck, W.: An architectural pattern language of cloud-based applications. In: PLoP 2011. ACM, October 2011
20. Fehling, C., Leymann, F., Rütschlin, J., Schumm, D.: Pattern-based development and management of cloud applications. Future Internet **4**(1), 110–141 (2012)
21. Großmann, M., et al.: Efficiently managing context information for large-scale scenarios. In: PerCom 2005. IEEE (2005)
22. Henricksen, K., Indulska, J.: A software engineering framework for context-aware pervasive computing. In: PerCom 2004. IEEE (2004)
23. Judd, G., Steenkiste, P.: Providing contextual information to pervasive computing applications. In: PerCom 2003. IEEE (2003)
24. Keller, A., Hellerstein, J.L., Wolf, J.L., Wu, K.L., Krishnan, V.: The CHAMPS system: change management with planning and scheduling. In: NOMS 2004, pp. 395–408. IEEE (2004)
25. Leymann, F.: Cloud computing: the next revolution in IT. In: The Photogrammetric Record, pp. 3–12, September 2009
26. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall PTR, USA (2000)
27. Lu, H., Shtern, M., Simmons, B., Smit, M., Litoiu, M.: Pattern-based deployment service for next generation clouds. In: SERVICES 2013, pp. 464–471. IEEE, June 2013
28. El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., Konstantinou, A.V.: Model driven provisioning: bridging the gap between declarative object models and procedural provisioning tools. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, pp. 404–423. Springer, Heidelberg (2006)
29. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0, May 2013
30. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: USITS. USENIX Association, June 2003
31. Roman, M., Campbell, R.H.: Gaia: enabling active spaces. In: SIGOPS 2000, pp. 229–234. ACM (2000)
32. Scheibenberger, K., Pansa, I.: Modelling dependencies of it infrastructure elements. In: BDIM 2008, pp. 112–113. IEEE, April 2008