Context-aware Cloud Application Management

Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann and Matthias Wieland
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany

Keywords: Application Management, Context, Automation, Cloud Computing.

Abstract: The automation of application 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 higher risk of unexpected 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 single components 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 management tasks in advance by analyzing the context in which tasks are executed. We validate the method practically and show how context-aware expert management knowledge can be applied fully automatically to running Cloud applications.

1 INTRODUCTION

In recent years, Cloud Computing gained a lot of attention due to its economical and technical benefits. Especially properties such as pay-on-demand computing, self-service, and elasticity enable enterprises to outsource IT (Leymann, 2009). These properties are key success factors for both Cloud providers as well as customers. To exploit these properties reliably for their offerings, Cloud providers have to automate their internal processes for application provisioning, configuration, and management. Therefore, a lot of tools, methods, and technologies have been developed in the past years: Script-centric DevOps communities support automated configuration management, guidelines such as the Information Technology Infrastructure Library (ITIL) (Malone et al., 2008) help applying best practices to IT service management, and standards such as TOSCA (OASIS, 2013) enable application developers to describe applications and their management in a portable way—to name a few examples. However, automating management tasks on complex Cloud applications becomes a more and more difficult challenge: The increasing number and diversity of employed Cloud services on various layers offered by different Cloud providers leads to a higher risk of unexpected side effects when they are combined with each other or integrated with traditional software components and middleware systems. Additional complexity arises from the heterogeneity of management technologies: Cloud providers, middleware systems, and software components often provide proprietary management APIs, security mechanisms, and data formats (Breitenbücher et al., 2013b). Thus, if a task affects multiple different parts of an application, also the management technologies of these parts may need to be integrated. This requires (i) a deep technical insight in each part and management technology as well as (ii) an overall understanding of the whole system and possible impacts to avoid errors. In these scenarios, often only experts of the respective technology are able to execute management tasks correctly. However, also experts reach their limits when a management task has to be executed on a complex application whose exact structure and runtime information are not documented: Unknown relations and dependencies between components that directly influence each other's functionality lead to a serious management challenge. Even if only a small operation has to be executed on a single component, its impact on other components is not predictable seriously if the application's structure is not known exactly. Thus, if the *context*, in which a management task is executed, is not known, understood, and considered, there is a high risk of unexpected side effects that may compromise the application's functionality. In addition, as executing management task manually is slow, costly, and error prone, Cloud application management must be automated to enable Cloud properties (Fehling et al., 2012; Oppenheimer et al., 2003).

Breitenbücher U., Binz T., Kopp O., Leymann F. and Wieland M..
Context-aware Cloud Application Management.
DOI: 10.5220/0004949204990509
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 499-509
ISBN: 978-989-758-019-2

As a result, in case multiple heterogeneous Cloud services, software components, and middleware systems are involved and different management technologies have to be orchestrated, it's of vital importance to (i) apply expert management knowledge, (ii) consider the context in which tasks are executed, and (iii) automate management processes to avoid manual errors.

In this paper, we tackle these issues. We present an approach that enables applying expert management knowledge for tasks in a certain context automatically to running Cloud applications. Therefore, we introduce an abstract (i) *Context-Aware Application Management Method* and (ii) present a fully automated realization of this method for Cloud Computing 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 required for many management tasks on Cloud applications due to the heterogeneous nature of the involved components and management technologies—which is not possible having traditional imperative models such as workflows or scripts as these models describe only the task-specific information but not the context in which they are executed. The presented automated realization of the method for Cloud application management validates the method's practical feasibility. It enables Cloud providers to offer a variety of different Cloud applications consisting of heterogeneous components without the need to employ or educate specialized experts that have the required technical management knowledge.

The remainder of this paper is structured as follows: In Section 2, we describe a motivating scenario to explain the method, which is introduced in Section 3. Section 4 presents the automated realization of the method to validate its practical feasibility for Cloud application management. In Section 5 we give an overview on related work. Section 6 concludes the paper and gives an outlook on future work.

2 MOTIVATION

In this section, we describe the problem tackled by our approach in detail and introduce a motivating scenario that is used throughout the paper to explain the presented method and its realization.

2.1 Problem Statement

In this section, we describe the major issues of Cloud application management and why context consideration is of vital importance in this area. Due to different functional as well as non-function requirements on services, Cloud offerings of different providers often have to be integrated as using a single Cloud provider for the complete application landscape of a company is often not possible (Fehling et al., 2012). The focus of this paper lies, therefore, on the management of *Complex Composite Cloud Applications*, which consist of multiple heterogeneous Cloud services, software components, and middleware systems of different vendors and Cloud providers.

The management of such applications is quite difficult as it requires a deep technical understanding of the involved components, the dependencies between them, and their management technologies—which are most often proprietary implementations providing heterogeneous management interfaces (Leonhardt, 2013). As a consequence, human-induced failures account for a significant number of outages as human errors accompany even the simplest system operation and management tasks (Brown and Patterson, 2001). The management flexibility recently introduced by Cloud Computing has additionally increased this effect as management tasks, such as the provisioning of new virtual machines, are literally at the systems manager's fingertips and often handled completely via proprietary management interfaces offered by Cloud providers (Fehling et al., 2012). Due to this management evolution, human errors are more likely to occur since the physical infrastructure and thereon deployed software systems are often virtualized and, therefore, obfuscating the actual structure of an application. Thus, it becomes more and more difficult to consider the *context*, in which management tasks are executed, correctly as application structures, system boundaries, and dependencies are becoming increasingly blurred by the Cloud. In addition, due to the lack of interoperability between Cloud services of different providers and traditional software components, most often complex configuration management is needed to integrate heterogeneous systems. As a result, executing management tasks which affect multiple parts of an application at once requires special management expertise. These problems impede automating Cloud application management, which is a major requirement for efficient use of Cloud resources and to avoid manual human errors (Fehling et al., 2012; Oppenheimer et al., 2003). Therefore, we need a means to capture and apply context-aware management expertise fully automatically to running Cloud applications.

2.2 Management Automation

To automate application management, the execution of management tasks is often described imperatively using executable processes implemented as workflows (Leymann and Roller, 2000) in languages such as BPEL (OASIS, 2007) or BPMN (OMG, 2011), scripts such as bash, or programs implemented in a computer programming language such as Java. Especially the more and more emerging script-centric configuration management technologies such as Chef (Nelson-Smith, 2013) are prime examples for this kind of imperative management descriptions. However, if an application is crucial to the business of an enterprise, errors that possibly result in system downtime are not acceptable. Therefore, before executing such management processes, they must be verified by experts to ensure correctness. Unfortunately, one of the most important problems of this approach is that 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, 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 process replacing the database through installing the required new 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 tasks and the context.

2.3 Motivating Scenario

In this section, we describe the motivating scenario that is used throughout the paper 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 should be migrated from one Cloud to another. Because the application evolved over time, it is currently distributed over two Clouds: The PHP frontend is hosted on Microsoft's public Cloud offering "Windows Azure"[1], the PostgreSQL database on Amazon's PaaS offering "Relational Database Service (RDS)"[2], which enables hosting databases directly as a service. The PHP frontend runs on an Apache HTTP Server (including the 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)"[3] in order 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 by experts having the knowledge to recognize the following two problems 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 on it. 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 Moodle 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 on 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 cannot connect to the database. The reason for both problems lies in ignoring the context in which the tasks are executed: First, if an application shall connect to a certain type of database, the runtime environment hosting the

[1] http://www.windowsazure.com/

[2] http://aws.amazon.com/rds/

[3] http://aws.amazon.com/ec2/

application must support connecting to this kind of database. It is not enough to simply set the configuration as the underlying infrastructure, i. e., the context, needs to be considered, too. Second, accessing a database hosted on Amazon RDS requires also more than simply writing endpoint information into a configuration file as also the firewall of the service has to be configured. Thus, also for this task, the context in the form of the infrastructure that hosts the database has to be considered to recognize the problem.

However, both problems cannot be detected by experts if the migration is implemented using traditional approaches such as management workflows or scripts: The processes would only model 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, username, and password in the frontend's configuration. The process neither provides information about the database's type nor which infrastructure is used to host the database—the context is not described. In addition, if administrators execute this migration manually, they require a lot of technical expertise and background information about the different APIs and employed technologies. Thus, we need a means to (i) capture expert management knowledge for a certain context and (ii) make it applicable fully automatically in order to help administrators avoiding unintended mistakes.

3 CONTEXT-AWARE MANAGEMENT METHOD

In this section, we introduce the *Context-Aware Application Management Method* that provides a means to consider the context in which management tasks on application components or relationships are executed. The method separates between a declarative description of the management tasks to be executed and the final executable management process. It consists of six steps which are shown in Figure 1: First, (1) we capture all runtime information and the structure of the application to be managed in a formal model which is extended in the second step (2) by declaring tasks on components and relations. In the third step, (3) experts analyze this model for unintended side effects and (4) adapt the model afterwards if necessary to resolve the found problems. In the fifth step, (5) this declarative model is translated into an imperative process model that is executed in the last step (6) to perform the tasks on the running application. We explain each step in the following subsections and start

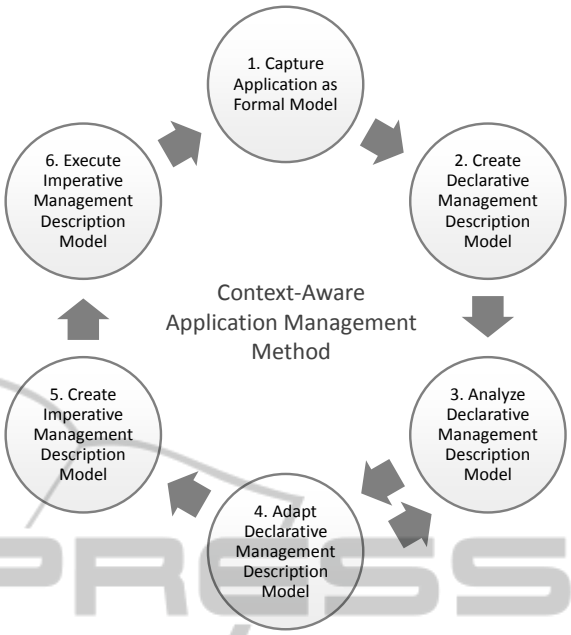

Figure 1: Context-Aware Application Management Method.

each description with a short summary of the step in italic text to provide a compact overview that is detailed afterwards.

3.1 Step 1: Capture Application as Formal Model

Describe all components of the application and their relations in a formal model that specifies their types and current runtime information.

First, the application to be managed is described as a formal model. This model captures the application structure and its state, i. e., all *components* such as Web servers, virtual machines, or installed applications, the *relations* between them, e. g., database connections, and their *runtime information*. The semantics of components and relations (we call both *model elements* in the following) 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 model elements, types can be inherited: the "ApacheHTTPServer" type is a subtype of "HTTPServer". Runtime information are described as model element properties, e. g., the "ApacheHTTPServer" has properties "IP-Address" and "Port" that specify the server's endpoint. The property schema is defined by the type of the model element. This formalization of the running applica-

tion provides a detailed, structured, and machine readable means to document a current snapshot of the application structure and all runtime information.

3.2 Step 2: Create Declarative Management Description Model

Create a Declarative Management Description Model that declaratively describes the management tasks to be executed on components and relations.

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 of the application. 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 a relation of type "SQLConnection" between a PHP application and an 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 tasks.

3.3 Step 3: Analyze Declarative Management Description Model

Analyze the impact of tasks in consideration of the context defined by the components, relations, and runtime information described by the DMDM.

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 adjacent relations of the application that are possibly 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: Experts on the "ApacheHTTPServer" are able to detect that the installation of a certain database connector is required, experts on 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.

3.4 Step 4: Adapt Declarative Management Description Model

Adapt the analyzed Declarative Management Description Model if necessary to solve the analyzed problems.

After the expert analysis, found problems have to be resolved in order to achieve the desired management goals. Therefore, the DMDM is adapted in this step by the respective experts to enable correct execution of the management tasks. Therefore, 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 changed by 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 by management experts.

3.5 Step 5: Create Imperative Management Description Model

Create an Imperative Management Description Model in the form of an executable process to perform the management tasks declared in the DMDM.

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 describes how the tasks have to be executed *imperatively*, 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.

3.6 Step 6: Execute Imperative Management Description Model

Execute the created Imperative Management Description Model to manage the running application.

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. Afterwards, the method may start from the beginning to execute further tasks.

4 VALIDATION

The presented method enables combining *declarative* management descriptions, which include all relevant context information to verify the management 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 a *theoretic* basis for enabling automated context-aware application management. In this section, we validate the proposed method by showing a fully automated *practical* implementation using existing frameworks. We describe the realization of the prototype for all steps of the method in the following.

4.1 Formalizing Applications Using Enterprise Topology Graphs

In Step 1, the application's structure and runtime information needs to be captured in a formal model. We use *Enterprise Topology Graphs (ETG)* (Binz et al., 2012) as model language as they are a common way to capture such information formally. ETGs are directed, possibly acyclic, graphs that describe the application's structure as topology model that contains each component as typed node and each relationship as typed edge between the nodes. Runtime information of nodes and relations are captured as properties of the respective 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. Figure 2 shows the motivating scenario as ETG example, graphically rendered using the visual topology notation "Vino4TOSCA" (Breitenbücher et al., 2012). Each component is depicted as rounded rectangle, relations as arrows between these rectangles, and runtime information as key-value properties. Element IDs are undecorated text, element types are enclosed by parentheses. Binz et al. showed that ETGs of running applications can be discovered fully automatically using the *ETG Discovery Framework* (Binz et al., 2013). Thus, the first step of formalizing the application to be managed can be automated by using this framework.

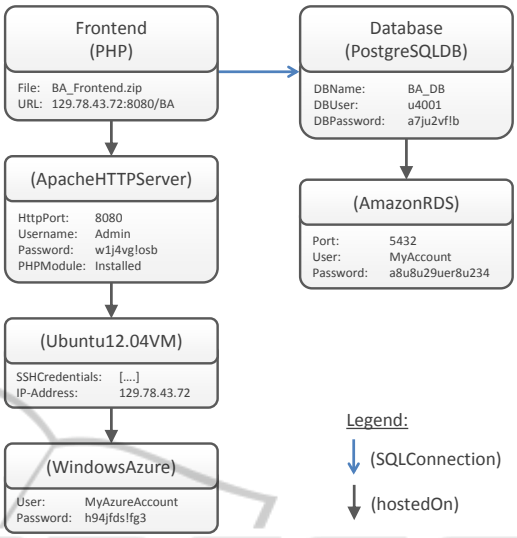

Figure 2: ETG of the running application.

4.2 Automating DMDM Creation Using Automated Management Patterns

Capturing running application snapshots in such formal ETG models provides a structured means to describe the context in which a management task is executed. Therefore, to create the DMDM in the second step, 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. (Breitenbücher et al., 2013a), 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 3 shows the Desired Application State Model that describes our migration motivating scenario. The colored circles with the symbols inside represent the management tasks to be executed in the

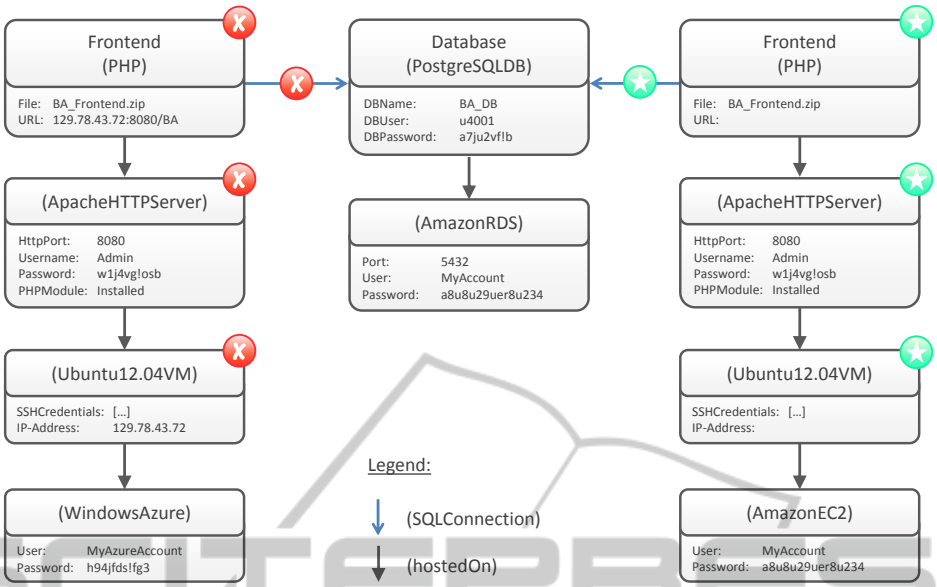

Figure 3: Desired Application State Model resulting from applying the Automated Management Pattern to the discovered ETG.

form of so-called *Management Annotations* (Breitenbücher et al., 2013a). 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 element it is attached to has to be created, while the red colored "Destroy-Annotations" with the "x" inside declare that the model element has 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 (Breitenbücher et al., 2013c).

Annotating management tasks to ETGs, i. e., creating a DMDM, can be automated, too: Desired Application State Models can be created automatically by applying so-called *Automated Management Patterns* to ETGs (Breitenbücher et al., 2013a). An Automated Management Pattern captures management expertise through implementing a transformation that annotates management tasks in the form of Management Annotations fully automatically to an input ETG. This transformation attaches, configures, or removes nodes, relations, and Management Annotations. In addition, Automated Management Patterns can be configured via input parameters. For example, the Desired Application State Model shown in Figure 3 is the result of applying the "Migrate PHP-based Application to Amazon Pattern", which was configured to use Amazon's IaaS offering EC2 for hosting the PHP frontend. The only manual step is selecting and configuring the pattern. Thus, Step 2 can be automated, too.

4.3 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 at 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 to (i) analyze correctness and (ii) may provide expert management knowledge required to adapt the model if necessary. The fragment is used for matchmaking of CAMTAs and Desired State Models: If all elements in a CAMTA's

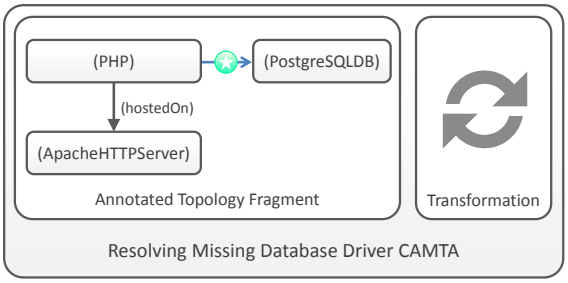

Figure 4: CAMTA that recognizes the problem of missing PostgreSQL database connector driver.

fragment match elements 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 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 State Model. Figure 4 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 SQL-Connection in the context of a PHP Application running on the Apache HTTP Server to a PostgreSQL database is possible. This is expressed by its Topology Fragment on the left. The transformation shown on the right analyzes the Desired State Model and finds out that the PostgreSQL connector driver is missing. Therefore, it adds the corresponding elements and tasks to the model. Thus, based on two CAMTAs, the Desired State Model, which results from applying the automated migration pattern in Step 2, gets adapted fully automatically for resolving the issues of the missing database connector and Security Group configuration. Therefore, the respective CAMTAs insert two different Management Annotations into the Desired State Model: (i) a "ConfigureSecurityGroup-Annotation" that is attached to the AmazonRDS node and an "InstallDriver-Annotation" attached to the Apache HTTP Server node. The ConfigureSecurityGroup-Annotation configures the AmazonRDS 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.

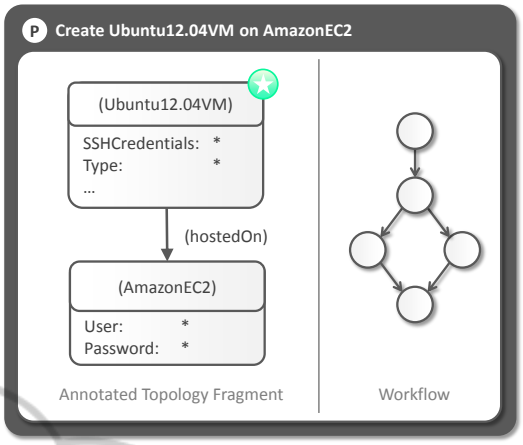

Figure 5: Management Planlet that creates an Ubuntu12.04 virtual machine on Amazon EC2.

As Desired State Models typically specify multiple management 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 add or configure elements, all CAMTAs need to be applied every time after one transformed the model to ensure that all tasks are validated in the current context that may be changed by formerly applied CAMTAs. As soon as input and output model do not change anymore after applying all matching CAMTAs, Step 4 is finished. The approach is similar to Automated Management Patterns (Breitenbücher et al., 2013a) that also use transformations to modify models.

4.4 Management Plan Generation

After the DMDM was analyzed and adapted for correctness in Step 3, the resulting model is not executable directly as it describes the tasks to be performed only declaratively, i. e., without implementation or control flow: The Declarative Management Description Model has to be transformed into an executable Imperative Management Description Model in Step 5. Therefore, we employ the management framework presented by Breitenbücher et al. (Breitenbücher et al., 2013a) that employs *Management Planlets* to translate Desired Application State Models fully automatically into executable BPEL workflows. Management Planlets 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 (Breitenbücher et al., 2013c). They serve as generic man-

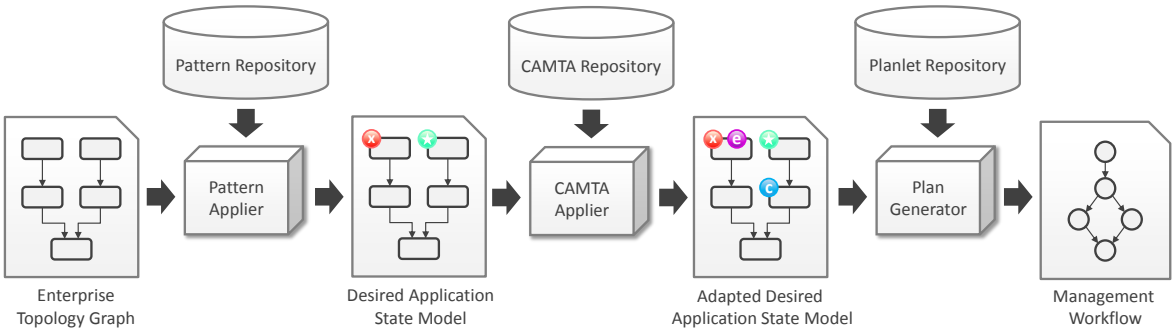

Figure 6: Conceptual Architecture of the method's realization.

agement 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 specified context. For example, the Planlet shown in Figure 5 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 executable workflow implements exactly the management logic required to create this virtual machine on EC2. Based on these fragments, different Planlets can be orchestrated to implement an overall management plan that performs all management tasks defined in the Desired Application State Model. Therefore, the framework employs a Plan Generator that transforms Desired Application State Models into executable management workflows. The whole method realization is shown in Figure 6: A discovered ETG is transformed into a Desired Application State Model by applying an Automated Management Pattern. This model is analyzed and possibly adapted by several CAMTAs. The Plan Generator gets the resulting model as input, searches Planlets that are able to execute the specified Management Annotations, and orchestrates the matching Planlets' workflows to an overall management workflow. This workflow implements all management tasks declared by the Desired Application State Model and can be executed in Step 6 to perform the tasks on the real running application.

4.5 Evaluation of the Realization

The introduced declarative model layer describing tasks in its formal context that encompasses all application components, their relations, and runtime information enables experts to detect dependencies that are not visible in traditional imperative management processes such as workflows. The reason is that these traditional processes describe only logic for the directly affected components, not the surrounding environment context: Details of the application structure, i. e., the context, are not contained in such process models and can, thus, not be analyzed. However, especially workflow technology and script-centric technologies such as Chef are well-established technologies used in systems management due to the provided features and properties: Workflow technology provides features, such as recoverability and compensation mechanisms, that enable correct and complete execution of long running processes, which is often a major requirement (Leymann and Roller, 2000). On the other side, the script-centric DevOps communities provide a lot of templates and implementations that can be used for configuration management without further modifications. The method enables using these technologies for the actual execution of the management tasks in the form of IMDMs. Thus, it enables combining both types of management description models to support both consideration of context and profiting from features provided by common technologies.

The presented realization is limited in terms of completeness: Management tasks in a Desired Application State Model are analyzed and possibly adapted only if there are CAMTAs that provide the required knowledge for analyzing and adapting the tasks in the actual context. The approach could be extended by marking all verified management tasks, i. e., all Management Annotations that were considered by one or more CAMTAs. In addition, a description of the task analysis could be added by CAMTAs to provide information about the analyzed issues and determined problems. This may help users to understand the performed analysis and enables drawing their attention to the tasks that were not considered by the system. These tasks could be then analyzed manually by experts before generating the final management workflow.

5 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 (Dey et al., 2000): "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 (Dey et al., 2001) and systems that support smart environments like Aura (Judd and Steenkiste, 2003) or Gaia (Roman and Campbell, 2000). Different types of development frameworks, e. g., the framework of Henricksen and Indulska (Henricksen and Indulska, 2004), and context management platforms, e. g., the Nexus Platform (Großmann et al., 2005), 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 topology models. The Declarative Management Description Model introduced in this paper provides a kind of Context Model that (i) enables to capture the environment in which management tasks are executed and (ii) the management tasks themselves described in a declarative fashion. 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 (Binz et al., 2013). 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 (Scheibenberger and Pansa, 2008) 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) (Distributed Management Task Force, 2010) 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. (Eilam et al., 2011), Maghraoui et al. (Maghraoui et al., 2006), and Keller et al. (Keller et al., 2004). The first two focus mainly on provisioning of applications whereas the third also considers application management. In general, the proposed method can be applied to approaches and frameworks 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 (Breitenbücher et al., 2014), we showed how declarative provisioning descriptions can be transformed automatically into imperative provisioning workflows based on the TOSCA standard (OASIS, 2013). The application to be provisioned is described as a topology, which models all components and relations of the application. As the tasks to be executed are clear (create each application component and instantiate the relations between the components) and the whole context of the provisioning is provided by this model in the form of the application topology, the presented method can be applied to this standards-based provisioning approach, too.

6 CONCLUSIONS

In this paper, we introduced an abstract Context-Aware Application Management Method that enables applying context-aware management expertise to running Cloud applications. We showed that separat-

ing models for context-aware analysis and management 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 Cloud Application Management using existing frameworks and technologies. In future, 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

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2013). Automated Discovery and Maintenance of Enterprise Topology Graphs. In *SOCA*, pages 126–134. IEEE.

Binz, T., Fehling, C., Leymann, F., Nowak, A., and Schumm, D. (2012). Formalizing the Cloud through Enterprise Topology Graphs. In *CLOUD*, pages 742–749. IEEE.

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

Breitenbücher, U., Binz, T., Kopp, O., and Leymann, F. (2013a). Pattern-based Runtime Management of Composite Cloud Applications. In *CLOSER*. SciTePress Digital Library.

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

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wieland, M. (2013c). Policy-Aware Provisioning of Cloud Applications. In *SECURWARE*, pages 86–95. Xpert Publishing Services.

Breitenbücher, U. et al. (2012). Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In *CoopIS*, pages 416–424. Springer.

Brown, A. B. and Patterson, D. A. (2001). To err is human. In *EASY*, page 5.

Dey, A. K., Abowd, G., and Salber, D. (2000). A context-based infrastructure for smart environments. In *Managing Interactions in Smart Environments*, pages 114–128. Springer.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166.

Distributed Management Task Force (2010). Common Information Model.

Eilam, T. et al. (2011). Pattern-based composite application deployment. In *Integrated Network Management*. IEEE.

Fehling, C., Leymann, F., Rütschlin, J., and Schumm, D. (2012). Pattern-based development and management of cloud applications. *Future Internet*, 4:110–141.

Großmann, M., Bauer, M., Hönle, N., Käppeler, U.-P., Nicklas, D., and Schwarz, T. (2005). Efficiently Managing Context Information for Large-Scale Scenarios. In *PerCom*. IEEE.

Henricksen, K. and Indulska, J. (2004). A Software Engineering Framework for Context-Aware Pervasive Computing. In *PerCom*. IEEE.

Judd, G. and Steenkiste, P. (2003). Providing contextual information to pervasive computing applications. In *PerCom*. IEEE.

Keller, A., Hellerstein, J. L., Wolf, J. L., Wu, K.-L., and Krishnan, V. (2004). The champs system: change management with planning and scheduling. In *NOMS*, pages 395–408. IEEE.

Leonhardt, S. (2013). A generic artifact-driven approach for provisioning, configuring, and managing infrastructure resources in the cloud. Diploma thesis, University of Stuttgart, Germany.

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

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

Maghraoui, K. E. et al. (2006). Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools. In *Middleware*, pages 404–423. Springer.

Malone, T., Blokdijk, G., and Wedemeyer, M. (2008). *ITIL V3 Foundation Complete Certification Kit*. Art of Service Pty Limited.

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

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

Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). Why do internet services fail, and what can be done about it? In *USITS*. USENIX Association.

Roman, M. and Campbell, R. H. (2000). Gaia: Enabling active spaces. In *SIGOPS*, pages 229–234. ACM.

Scheibenberger, K. and Pansa, I. (2008). Modelling dependencies of it infrastructure elements. In *BDIM*, pages 112–113. IEEE.