Automating Cloud Application Management Using Management Idioms

Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart, Stuttgart, Germany
{breitenbuecher, lastname}@iaas.uni-stuttgart.de

***Abstract*—Patterns are a well-established concept to document generic solutions for recurring problems in an abstract manner. Especially in Information Technology (IT), many pattern languages exist that ease creating application architectures, designs, and management processes. Their generic nature provides a powerful means to describe knowledge in an abstract fashion that can be reused and refined for concrete use cases. However, the required manual refinement currently prevents applying the concept of patterns efficiently in the domain of Cloud Application Management as automation is one of the most important requirements in Cloud Computing. This paper presents an approach that enables automating both (i) the refinement of management patterns for individual use cases and (ii) the execution of the refined solutions: we introduce Automated Management Idioms to refine patterns automatically and extend an existing management framework to generate executable management workflows based on these refinements. We validate the presented approach by a prototypical implementation to prove its technical feasibility and evaluate its extensibility, standards compliance, and complexity.**

***Keywords*—Application Management; Automation; Patterns; Idioms; Cloud Computing**

I. Introduction

Patterns are a well-established concept to document reusable solution expertise for frequently recurring problems. In many areas, they provide the basis for decision making processes, design evaluations, and architectural issues. In the domain of Cloud Computing, patterns are of vital importance to build, manage, and optimize IT on various levels: Cloud Computing Architecture and Management Patterns [1], Enterprise Integration Patterns [2], and Green IT Patterns [3] are a few examples that provide helpful guides for realizing complex Cloud applications, their management, and challenging non-functional requirements. The concept of patterns enables IT experts to document knowledge about proven solutions for problems in a certain context in an abstract, structured, and reusable fashion that supports systems architects, developers, administrators, and operators in solving concrete problems. The abstract nature of patterns enables generalizing the core of problem and solution to a level of abstraction that makes them applicable to various concrete instances of the general problem—independently from individual manifestations. Thus, many IT patterns are applicable to a big set of different settings, technologies, system architectures, and designs. Applying patterns to real problems requires, therefore, typically a *manual refinement* of the described abstract high-level solution for adapting it to the concrete use case. To guide this refinement and ease pattern application, most pattern languages document "implementations", "known uses", or "examples" for the described pattern solution—as already done implicitly by Alexander in his early publications on patterns [4].

However, in some areas, these benefits face difficulties that decrease the efficiency of using patterns immensely. In the domain of *Cloud Application Management*, the immediate, fast, and correct execution of management tasks is of vital importance to achieve Cloud properties such as on-demand self-service and elasticity [5][6]. Thus, management patterns, e. g., to scale a Cloud application, cannot be applied *manually* by human operators when a problem occurs because manual executions are too slow and error prone [1][7]. Therefore, to use patterns in Cloud Application Management, their application must be automated [8]. A common way to automate the execution of management patterns is creating executable processes, e. g., workflows [9] or scripts, that implement a refined pattern solution for a certain application [1]. To achieve Cloud properties, this must be done *in advance*, i. e., before the problem occurs, for being ready to run them immediately when needed. However, these processes are tightly coupled to a single application as the refinement of a management pattern depends mainly on the technical details of the application, its structure, and the concrete desired solution [10]. For example, the pattern for scaling a Cloud application results in different processes depending on the Cloud provider hosting the application: due to the heterogeneous management APIs offered by different Cloud providers, different operations have to be executed to achieve the same conceptual effect [11]. Thus, individual pattern refinement influences the resulting management processes fundamentally. As a result, multiple individual processes have to be implemented in advance to execute one management pattern on different applications. However, if multiple patterns have to be implemented for hundreds of applications in advance, this is not *efficient* as the required effort is immense. In addition, as Cloud application structures typically evolve over time, e. g., caused by scaling, a pattern may has to be implemented multiple times for a single application. Ironically, if the implemented processes are not used during the application's lifetime, the spent effort was laborious, costly, but completely unnecessary.

The result of the discussion above is that the gap between a pattern's abstract solution and its refined executable implementation for a certain use case currently prevents applying the concept of patterns efficiently to the domain of Cloud Application Management—due to the mandatory requirement of automation and its difficult realization. Thus, we need a means to automate the refinement of abstract patterns to concrete executable solutions on demand. In this paper, we tackle these issues by presenting an approach that enables applying management patterns fully automatically to individual applications. We show how the required refinement of a management pattern towards a concrete use case can be automated by inserting an additional layer of *Automated Management Idioms*, which provide a fine grained refinement of a particular abstract management pattern. These Automated Management Idioms are able to create formal declarative descriptions of the management tasks to be performed automatically for individual applications that are used afterwards to generate the corresponding executable management processes using an existing management framework. This enables automating the application of abstract patterns to various concrete applications without human intervention, which increases the efficiency of using patterns in Cloud Application Management. Thereby, the presented approach (i) helps IT experts to capture their management knowledge in an executable fashion and (ii) enables automating *existing* management patterns—both independently from individual applications. To prove the relevance of our approach, we conduct a detailed case study that illustrates the high complexity of applying an abstract migration management pattern to a concrete use case. We validate the approach through a prototype that extends an existing management framework by the presented concept and the implementation of real world migration use cases to prove its technical feasibility. Furthermore, we evaluate the concept in terms of automation, technical complexity, standards compliance, separation of concerns, and extensibility.

The paper is structured as follows: in Section II, we describe background information and a case study. In Section III, we describe the management framework which is extended by our approach presented in Section IV. In Section V, we evaluate the approach and present related work in Section VI. We conclude the paper and give an outlook on future work in Section VII.

II. Background, Motivation, and Requirements

In this section, we provide background information about the domain of Cloud Application Management and analyze the requirements to apply the concept of patterns. Afterwards, we motivate the presented approach through a case study.

A. *Automating Patterns for Cloud Application Management*

In this section, we discuss why the concept of patterns and their automation are of vital importance in the domain of Cloud Application Management to optimize management and to reduce the number of errors and downtimes. Due to the steadily increasing use of IT in enterprises, accurate operation and management are of crucial importance to align business and IT. As a consequence, requirements such as high-availability, elasticity, and cheap operation of IT arise increasingly. Therefore, more and more enterprises outsource their IT to external providers to achieve these properties reliably and to automate IT management—both enabled by Cloud Computing [12]. However, the fast growing number of proprietary, mostly heterogeneous, Cloud services offered by different providers leads to a high complexity of creating composite Cloud applications and executing management tasks as the (i) offered services, (ii) non-functional capabilities, and (iii) application programming interfaces (APIs) of different Cloud providers differ significantly from each other. As a result, the actual structure of an application, the involved types of Cloud services (e. g., platform services), and the management technologies to be used mainly depend on the providers and are hardly interoperable with each other due to proprietary characteristics. This results in completely different management processes for different Cloud providers implementing the same conceptual management task. As a consequence, the *conceptual solution* implemented in such processes gets obfuscated by technical details and proprietary idiosyncrasies. It is nearly impossible to reuse and adapt the implemented knowledge *efficiently* for other applications and providers as the required effort to analyze and transfer the conceptual solution to other use cases is much too high. This results in continually reinventing the wheel for problems that were already solved multiple times—but not documented in a way that enables reusing the conceptual knowledge. A solution for these issues are management patterns, which provide a well-established means to document conceptual solutions for frequently recurring problems in a structured, reusable, and tailorable way [13]. Therefore, a lot of architectural and management knowledge for Cloud applications and their integration was captured in the past years using patterns, e. g., Enterprise Integration Patterns [2] and Cloud Computing Architecture and Management Patterns [1]. A management pattern typically documents the general characteristics of a certain (i) problem, (ii) its context, and (iii) the solution in an abstract way without getting lost in technical realization details. Thereby, they help applying proven conceptual management expertise to individual applications, which typically requires a manual refinement of the pattern's abstract solution for the concrete use case. However, this manual refinement leads to two challenges that are tackled in this paper to enable applying the concept of management patterns efficiently in the domain of Cloud Application Management: (i) technical complexity and (ii) automation of refinement and solution execution.

1) Technical Complexity of Refinement: The technical layer of IT management and operation becomes a more and more difficult challenge as each new technology and its management issues increase the degree of complexity—especially if different heterogeneous technologies are integrated in complex systems [10]. Thus, the required refinement of a pattern's abstract solution to fine grained technical management operations as well as their correct parametrization and execution are complex tasks that require technical expert management knowledge.

2) Automation of Refinement and Solution Execution: In Cloud Computing, application management must be automated as the manual execution of management tasks is too slow and error-prone since human operator errors account for the largest fraction of failures in distributed systems [7][14]. Thus, the refinement of a pattern's abstract solution to a certain use case and the execution of the refined solution must be automated.

B. *Case Study and Motivating Scenario*

In this section, we present a case study that is used as motivating scenario throughout the paper to explain the approach and to illustrate the high complexity of applying abstract management patterns to concrete use cases. Due to the important issue of vendor lock-in in Cloud Computing, we choose a migration pattern to show the difficulties of refinement and the opportunities enabled by our approach. We explain the important details of the pattern and apply it to the use case of migrating a Java-based application to the Amazon Cloud. The selected pattern is called "Stateless Component Swapping Pattern" [15] and originates from the Cloud Computing pattern language developed by Fehling et al. [1][13][15]. The question answered by this pattern is *"How can stateless application components that must not experience downtime be migrated?"*. Its intent is extracting stateless application components from one environment and deploying them into another while they are

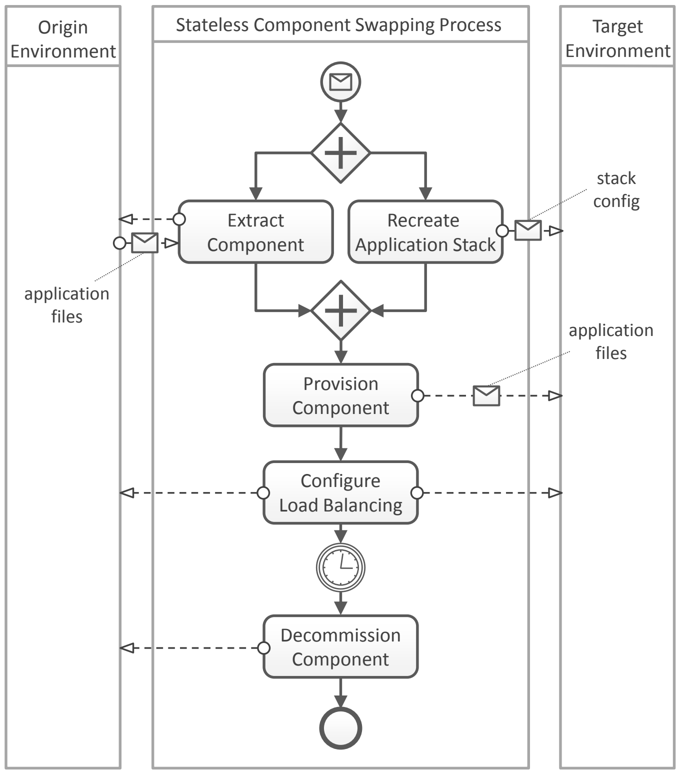

Figure 1. Abstract Stateless Component Swapping Process (adapted from [15]).

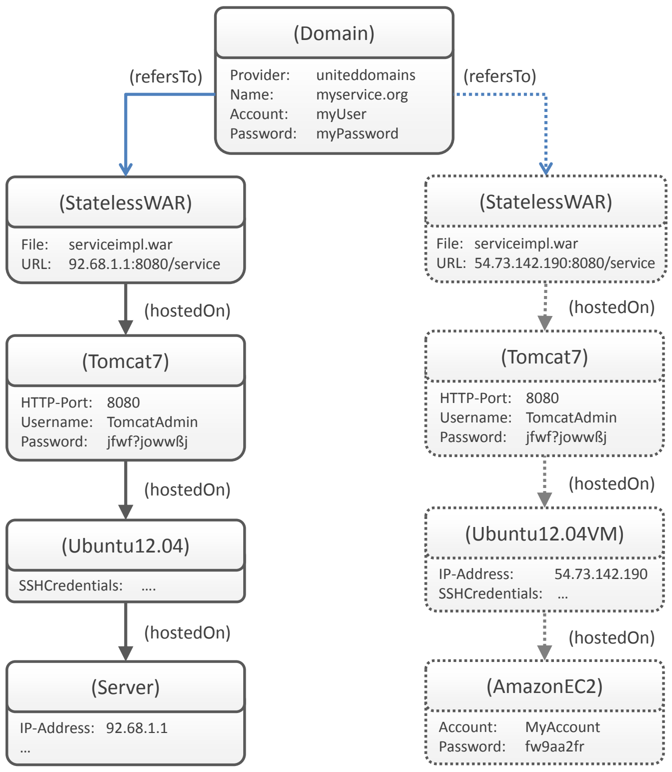

Figure 2. Use case for applying the Stateless Component Swapping Pattern.

active in both environments during the migration. Afterwards, the old components are decommissioned. The context observed by this pattern is that, in many business cases, the downtime of an application is unacceptable, e. g., for customer-facing services. A stateless application shall, therefore, be migrated transparently to the accessing human users or applications. Here, "stateless" means that the application does not handle internal session state: the state is provided with each request or kept in provider-supplied storage. Figure 1 depicts the pattern's solution as Business Process Model and Notation (BPMN) [16] diagram: the component to be migrated is first extracted from the origin environment while the required application stack is provisioned concurrently in the target environment. Then, the component is deployed on this stack in the target environment while the old component is still active. Finally, after the new component is provisioned, the reference to the migrated component is updated (load balancing) and the old component is decommissioned.

This pattern shall be applied to the following concrete use case. A stateless Webservice implemented in Java, packaged as Web Archive (WAR), is currently hosted on the local physical IT infrastructure of an enterprise. Therefore, a Tomcat 7 Servlet Container is employed that runs this WAR. The Tomcat is deployed on an Ubuntu Linux operating system that is hosted on a physical server. The Webservice is publicly reachable under a domain, which is registered at the domain provider "United Domains". This service shall be migrated to Amazon's public Cloud to relieve the enterprise's physical servers. Therefore, Amazon's public Cloud offering "Elastic Compute Cloud (EC2)" is selected as target environment. On EC2, a virtual machine with the same Ubuntu operating system and Tomcat version shall be provisioned to run the Webservice in the Cloud.

Figure 2 shows the technical details of this migration. On the left, the current Enterprise Topology Graph (ETG) [17] of the application is shown. An ETG is a formal model that captures the current state of an application in the form of a topology, which describes all components and relations including their types, configuration, and runtime information. ETGs of running applications can be discovered fully automatically using the "ETG Discovery Framework" [18]. We use the visual notation Vino4TOSCA [19] to render ETGs graphically: components are depicted as rounded rectangles containing their runtime properties below, relationships as arrows. Element types are enclosed by parentheses. On the right, the goal of the migration is shown: all components and relationships drawn with dotted lines belong to the goal state, i. e., the refined pattern solution.

To refine the pattern's abstract solution process shown in Figure 1 to this use case, the following tasks have to be performed: (i) the WAR to be migrated must be extracted from the origin environment, (ii) a virtual machine (VM) must be provisioned on EC2, (iii) a Tomcat 7 Servlet Container must be installed on this VM, (iv) the WAR must be deployed on the Tomcat, (v) the domain must be updated, and (vi) the old WAR must be decommissioned. The technical complexity of this migration is quite high as four different heterogeneous management APIs and technologies have to be combined and one workaround is required to achieve these goals: the extraction of the WAR deployed on the local Tomcat is not supported by Tomcat's management API [20]. Thus, a workaround is needed that extracts the WAR file directly from the underlying Ubuntu operating system. However, this is technically not trivial: an SSH connection to the operating system must be established, the respective directory must be found in which Tomcat stores the

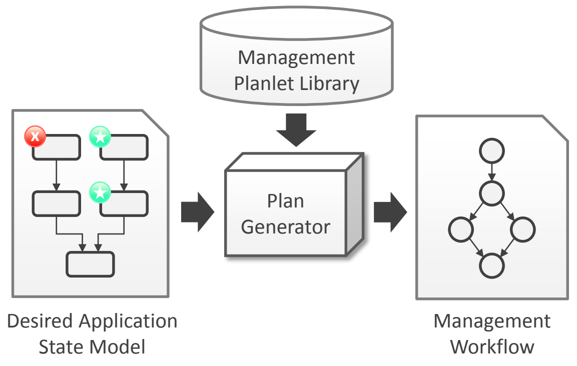

Figure 3. Management Planlet Framework Overview (adapted from [8]).

deployed WAR files, and the correct WAR must be transferred from the remote host to a local host using protocols such as Secure Copy (SCP). To provision a new virtual machine, EC2's HTTP-based management API [21] has to be invoked. However, an important security detail has to be considered here: per default, a virtual machine on EC2 is *not* accessible from the internet. Amazon employs so-called "Security Groups" to define firewall rules for accessing virtual machines. Thus, as the Webservice should be accessible from the internet, a Security Group must be defined to allow access. To install Tomcat 7, script-centric configuration management technologies such as Chef can be used. To deploy the WAR on Tomcat, Tomcat's HTTP-based management API has to be invoked. The domain can be updated using the management API of United Domains [22]. However, to avoid downtime, the old WAR must not be decommissioned until the Domain Name System (DNS) servers were updated with the new URL. Therefore, a DNS Propagation Checker must be employed. To summarize, the required technical knowledge to refine and implement the pattern's abstract solution for this concrete use case is immense. In addition, automating this process and orchestrating several management APIs that provide neither a uniform interface nor compatible data formats is complex, time-consuming, and costly as the process must be created by experts to avoid errors [10]. As a consequence, (i) handling the technical complexity and (ii) automating this process are difficult challenges.

III. EMPLOYED MANAGEMENT FRAMEWORK

The approach we present in this paper tackles these issues by extending the "Management Planlet Framework" [8][10][23]. This framework enables describing management tasks to be performed in an *abstract* and *declarative* manner using Desired Application State Models, which can be transformed fully automatically into executable workflows by a Plan Generator through orchestrating Management Planlets. In this section, we explain the framework briefly to provide all required information to understand the presented approach. The concept of the management framework is shown in Figure 3. The *Desired Application State Model (DASM)* on the left declaratively describes management tasks that have to be performed on nodes and relations of an application. It consists of (i) the application's ETG, which describes the current structure and runtime information of the application in the form of properties, and (ii) *Management Annotations*, which are declared on nodes and relations to specify the management tasks to be executed

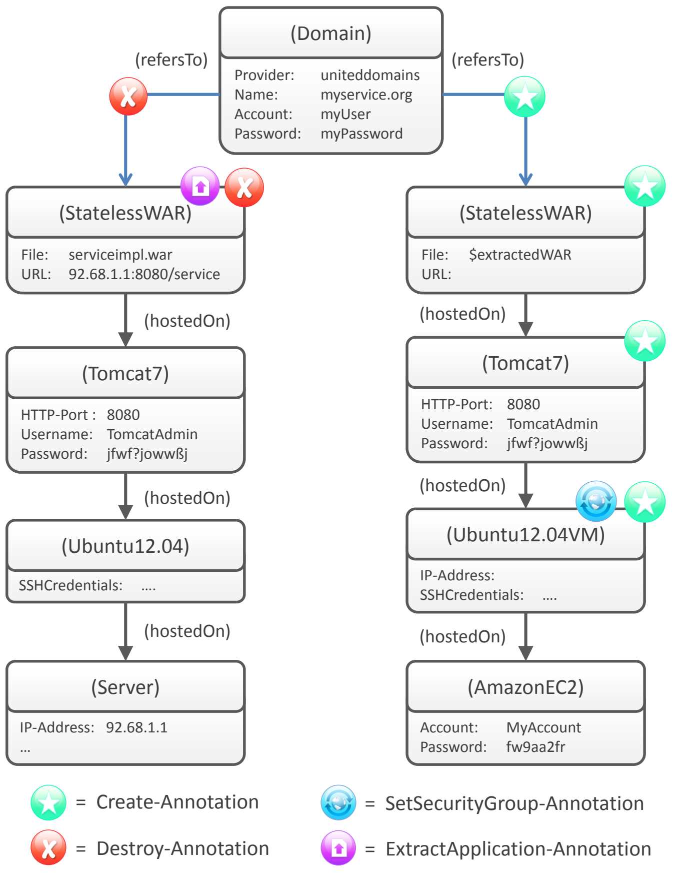

Figure 4. DASM describing the management tasks of the refined pattern.

on the associated element, e. g., to create, update, or destroy the corresponding node or relation. A Management Annotation (depicted as coloured circle) defines only the abstract semantics of the task, e. g., that a node should be created, but not its technical realization. Thus, in contrast to executable imperative management descriptions such as management workflows that define all technical details, a DASM describes the management tasks to be performed only declaratively, i. e., only the *what* is described, but not the *how*. As a consequence, DASMs are not executable and are, therefore, transformed into executable *Management Workflows* by the framework's *Plan Generator*.

Figure 4 shows a DASM that realizes the motivating scenario of Section II-B. The DASM describes the required additional nodes and relations as well as Management Annotations that declare the tasks to be performed: the green *Create-Annotations* with the star inside declare that the corresponding node or relation shall be created while the red circles with the "X" inside represent *Destroy-Annotations*, which declare that the associated element shall be destroyed. The magenta coloured *ExtractApplication-Annotation* is used to extract the application files of the WAR node, the blue coloured *SetSecurityGroup-Annotation* configures the Security Group to allow accessing the node from the internet. This DASM specifies the tasks described by the abstract solution of the Stateless Component Swapping Pattern refined to the concrete use case of migrating a Java Webservice to EC2. As Management Annotations describe tasks only declaratively, the model contains no technical details about management APIs, data formats, or the control flow.

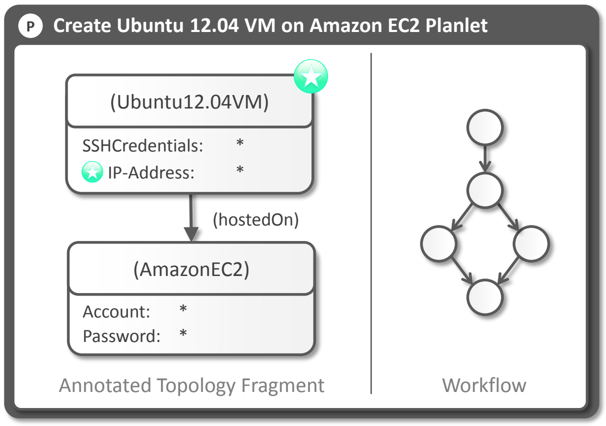

Figure 5. Management Planlet that creates an Ubuntu VM on Amazon EC2.

The framework's Plan Generator interprets DASMs and generates the corresponding executable workflows automatically. This is done by orchestrating so-called *Management Planlets*, which are workflows executing Management Annotations on nodes and relations such as deploying a WAR on Tomcat or updating a domain with a new URL. Planlets are developed by technology experts and provide the low-level imperative management logic to execute the declarative Management Annotations used in DASMs. Thus, they are reusable management building blocks implementing the *how* of the declaratively described abstract management tasks in DASMs. Management Planlets express their functionality through an *Annotated Topology Fragment*, which describes (i) the Planlet's *effects* in the form of Management Annotations it executes on elements and (ii) *preconditions* that must be fulfilled to execute the Planlet. The Plan Generator orchestrates suitable Planlets to process all Management Annotations in the DASM. The order of Management Planlets is determined based on their preconditions and effects: all preconditions of a Planlet must be fulfilled by the DASM itself or by another Planlet that is executed before.

Figure 5 shows a Planlet that creates a new Ubuntu 12.04 virtual machine on Amazon EC2. The Annotated Topology Fragment exposes the Planlet's functionality by a Create-Annotation attached to the node of type "Ubuntu12.04VM" which has a "hostedOn" relation to a node of type "AmazonEC2". The Planlet's preconditions are expressed by all properties that have no Create-Annotation attached: the desired "SSHCredentials" of the VM node as well as "Account" and "Password" of the Amazon EC2 node must exist to execute the Planlet, which takes this information to create the new VM (we omit other properties to simplify). The Planlet's effects on elements are expressed by Create-Annotations on their properties, i. e., the Create-Annotation on the "IP-Address" of the VM node means that the Planlet sets this property. The existence of this property and the "SSHCredentials" are typical preconditions of Planlets that install software on virtual machines. Thus, Planlets can be ordered based on such properties. The strength of Management Planlets is hiding the technical complexity completely [8]: the Plan Generator orchestrates Planlets based only on their Topology Fragments and the abstract Management Annotations declared in the DASM, i. e., without considering technical details implemented by the Planlet's workflow. This provides an abstraction layer to integrate management technologies.

IV. AUTOMATED REFINEMENT OF MANAGEMENT PATTERNS TO EXECUTABLE MANAGEMENT WORKFLOWS

In the previous section, we explained how DASMs can be used to describe management tasks in an abstract manner and how they are transformed automatically into executable workflows by the Management Planlet Framework. Thereby, the framework provides a powerful basis for automating patterns and handling the technical complexity of pattern refinement.

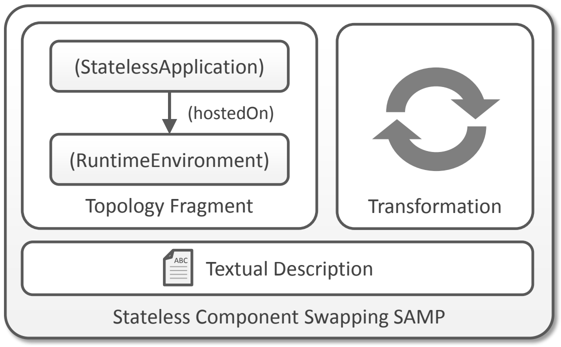

Figure 6. Stateless Component Swapping SAMP.

In a former work [8], we presented an approach to generate DASMs automatically by applying management patterns to ETGs of running applications. However, as patterns should not capture use case-specific information, the approach provides only a *semi-automated* means and requires a manual refinement of the resulting DASM. Consequently, we call these patterns *Semi-Automated Management Patterns (SAMP)*. In this paper, we extend the concept of SAMPs. Therefore, we introduce them briefly to provide required information. The input of a SAMP is the current Enterprise Topology Graph (ETG) of the application to which the pattern shall be applied, its output is a DASM that declaratively describes the pattern's solution in the form of Management Annotations to be performed. A SAMP consists of three parts, as shown in Figure 6: (i) Topology Fragment, (ii) Transformation, and (iii) textual description. The *Topology Fragment* is a small topology that defines the pattern's context, i. e., it is used to determine if a pattern is applicable to a certain ETG. Therefore, it describes the nodes and relations that must match elements in the ETG to apply the pattern to these matching elements. For example, the *Stateless Component Swapping SAMP* shown in Figure 6 is applicable to ETGs that contain a component of type "StatelessApplication" that is hosted on a component of type "RuntimeEnvironment". As the type "StatelessWAR" is a subtype of "StatelessApplication" and "Tomcat7" a subtype of "RuntimeEnvironment", the shown SAMP is applicable to the motivating scenario (cf. Figure 2). The second part is a *Transformation* that implements the pattern's solution logic, i. e., how to transform the input ETG to the output DASM that describes the tasks to be performed. However, SAMPs provide only a semi-automated means as the resulting DASM requires further manual refinement—similar to the required refinement of normal patterns for concrete use cases. Figure 7 shows the DASM resulting from applying the Stateless Component Swapping SAMP to the ETG described in the motivating scenario. As the pattern's transformation can not be aware of the desired refinement for this use case, it is only able to apply the *abstract* solution: (i) the stateless application is extracted,

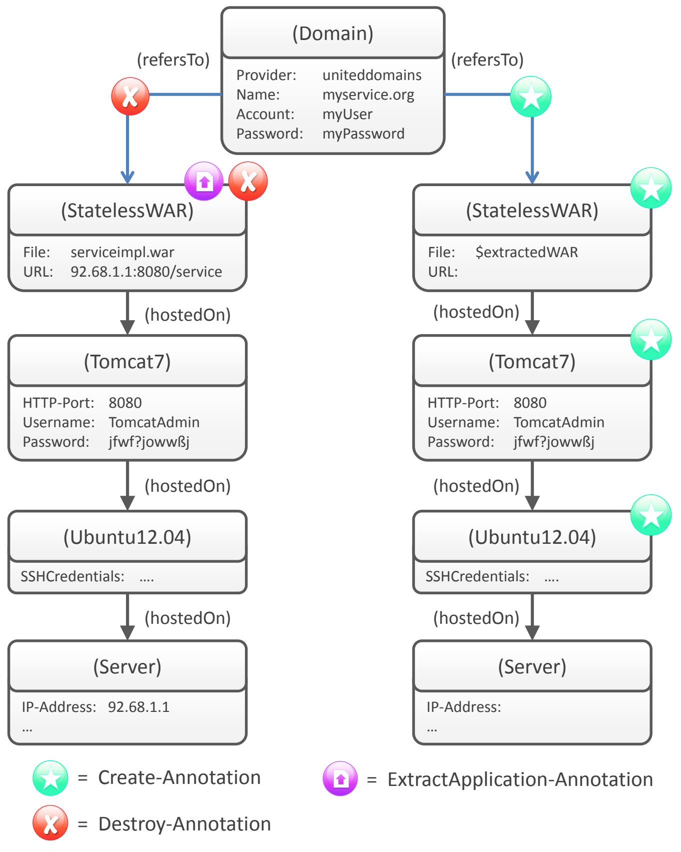

Figure 7. DASM after applying the Stateless Component Swapping SAMP.

(ii) the stack is copied, and (iii) all incoming relations of the application to be migrated are switched to the new deployment. To refine this DASM for the motivating scenario, the “Server” node type must be replaced by “AmazonEC2”, the operating system must be a virtual machine of type “Ubuntu12.04VM”, and a Management Annotation must be added to configure the Security Group (cf. Figure 4). However, manual refinement violates the two requirements analyzed in Section II-A as (i) the technical complexity remains (e. g., operator has to be aware of Security Groups) and (ii) human intervention is required, which breaks the required full automation. Of course, Semi-Automated Management Patterns could be created for concrete use cases as also shown in Breitenbücher et al. [8]. However, this violates the abstract nature of patterns and causes confusing dependencies between patterns and use cases, which decreases usability. In addition, with the increasing number of such use case-specific management patterns, the conceptual part of the abstract solution gets obfuscated. Therefore, we add an explicit refinement layer to separate the different layers of abstraction.

A. *Overview of the Approach*

In this and the next section, we present the main contribution of this paper. The approach enables applying patterns fully automatically to concrete use cases and solves the two problems analyzed in Section II-A: (i) handling the technical complexity of refinement and (ii) automating refinement and solution execution. The goal is to automate the whole process of applying a pattern selected by a human operator to a concrete use case. The reason that Semi-Automated Management Patterns can not be

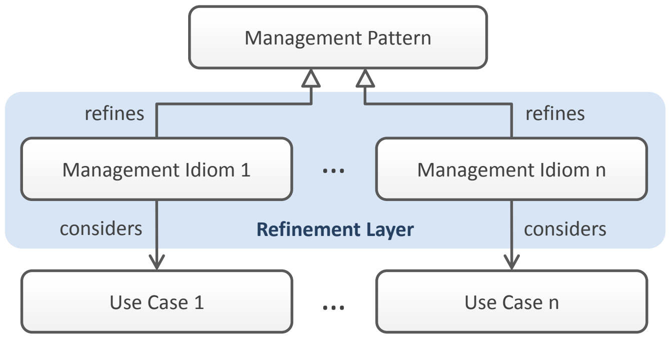

Figure 8. Additional refinement layer below management patterns.

applied fully automatically results from the generative nature of patterns since they describe only the general core of solutions. Consequently, they must be refined manually to individual use cases. Therefore, we need a more specific means below patterns to document concrete management problems and already refined solutions. According to Buschmann et al. [24], who consider design issues of software, so-called “idioms” represent *low-level patterns* that deal with the implementation of particular design issues in a certain programming language. They address aspects of both design and implementation and provide, thus, already refined pattern solutions. For example, an idiom may describe concretely how to implement the abstract *Model-View-Controller Pattern (MVC)* in Java. Thus, this concept solves a similar kind of refinement problem in the domain of software architecture and design. Therefore, we adapt this refinement concept to the domain of Cloud Application Management.

We insert an additional refinement layer below management patterns in the form of *Application Management Idioms*, which are a tailored incarnation of a certain abstract management pattern refined for a concrete problem, context, and solution of a certain use case. Figure 8 shows the conceptual approach: a management pattern is refined by one or more Management Idioms that consider each a concrete use case of the pattern and provide an already refined solution. Thus, instead of refining an abstract management pattern manually to a certain use case, the pattern to be applied can be refined automatically by selecting the appropriate Management Idiom directly. Applying this concept to our motivating scenario, the refinement of the Stateless Component Swapping Pattern to our concrete use case of migrating a stateless Java Webservice packaged as WAR to Amazon EC2 (see Section II-B) can be captured by a “Stateless WAR from Tomcat 7 to Amazon EC2 Swapping Idiom”, which describes the refined problem, context, and solution. Thus, instead of providing only the generic solution, this idiom is able to describe the tasks in detail tailored to this concrete context. Thereby, management tasks such as defining the Security Group can be described while the link to the conceptual solution captured by the actual pattern is preserved.

The additional refinement layer enables separating concerns: on the management pattern layer, the generic abstract problem, context, and solution are described to capture the *conceptual core*. On the Management Idiom layer, a *concrete refinement* is described that possibly obfuscates the conceptual solution partially to tackle concrete issues of individual use cases. As a consequence, the presented approach enables both (i) capturing generic management knowledge and (ii) providing tailored

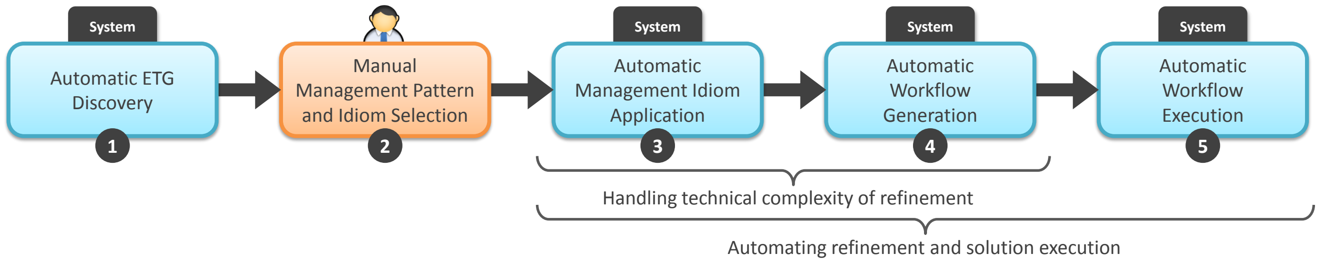
System
Automatic ETG Discovery
1
Manual Management Pattern and Idiom Selection
2
System
Automatic Management Idiom Application
3
System
Automatic Workflow Generation
4
System
Automatic Workflow Execution
5
Handling technical complexity of refinement
Automating refinement and solution execution

Figure 9. Process that describes how to apply a management pattern fully automatically to a running application using Automated Management Idioms.

refinements linking to the actual pattern. In the next section, we extend the concept of Semi-Automated Management Patterns by Automated Management Idioms following this concept.

B. Automated Management Idioms

To automate the new refinement layer, we introduce *Automated Management Idioms (AMIs)* in this section. Automated Management Idioms refine Semi-Automated Management Patterns through (i) providing a refined Topology Fragment that formalizes the particular context to which the idiom is applicable and (ii) implementing a more specific transformation tailored to the refined context for transforming the input ETG directly into an already refined DASM. Each AMI additionally defines a *Pattern Reference (PR)* linking to its original pattern.

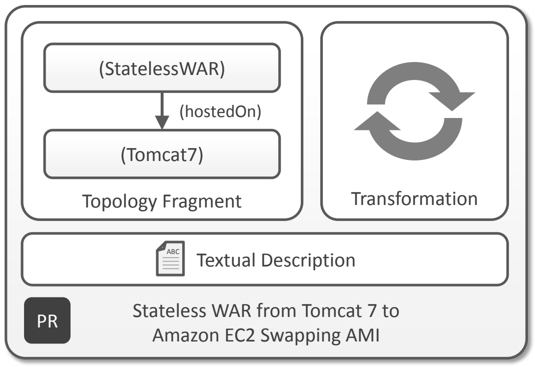

Figure 10. Stateless WAR from Tomcat 7 to Amazon EC2 Swapping AMI.

Figure 10 shows the AMI that refines the Stateless Component Swapping SAMP for our motivating scenario. The refinement of the context, to which this idiom is applicable, is expressed by the refined Topology Fragment: while the pattern is applicable to topologies that contain an abstract "StatelessApplication" that is hosted on an abstract "RuntimeEnvironment", the shown idiom refines this fragment and is only applicable to a node of type "StatelessWAR" that is hosted on a node of type "Tomcat7". The refinement of the Topology Fragment restricts the idiom's applicability and enables to implement a transformation that considers exclusively this concrete use case. Therefore, the idiom's transformation differs from the pattern's transformation by adding more details to the output DASM: it refines the "Server" node directly to "AmazonEC2" and adds the required Management Annotation that configures the Security Group accordingly in order to enable accessing the migrated WAR from the internet. The operating system is exchanged by "Ubtuntu12.04VM" while the "Tomcat7" and "StatelessWAR" nodes are simply copied from the ETG of the origin environment. Similar to the transformation of the pattern, all incoming relations of the old WAR node are redirected to the new WAR. Thus, the resulting DASM corresponds exactly to the DASM shown in Figure 4, which would be created manually by an expert to refine the pattern to this use case. As a result, applying this Automated Management Idiom to the motivating scenario's ETG results in a completely refined DASM which can be transformed directly into an executable workflow by the Management Planlet Framework. Thus, no further manual refinement is required to apply the pattern to this use case. Nevertheless, the Management Planlet Framework enables implementing this Automated Management Idiom on a high level of abstraction due to Management Annotations.

C. Fully Automated Pattern-based Management System

In this section, we explain the process of applying a management pattern in the form of an Automated Management Idiom automatically to a running application using the Management Planlet Framework. This process provides the basis for a *Fully Automated Pattern-based Management System* and consists of the five steps shown in Figure 9. First, the ETG of the application to be managed has to be discovered. This step is automated by using the ETG Discovery Framework [18], which gets an entry node of the application as input, e. g., the URL pointing to a deployed Webservice, and discovers the complete ETG of this application fully automatically including all runtime properties. In the second step, the user selects the pattern to be applied in the form of an Automated Management Idiom. This is the only manual step of the whole process. In the third step, the selected idiom is applied by the framework fully automatically to the discovered ETG. Therefore, the idiom's transformation is executed on the ETG. The output of this step is a Desired Application State Model that is based on the discovered ETG but additionally contains all management tasks to be executed in the form of Management Annotations. This DASM is used in Step 4 by the Management Planlet Framework to generate an executable management workflow, as described in Section III. In the last step, the generated workflow is executed using a workflow engine. The overall process fulfills the two requirements analyzed in Section II-A: (i) handling technical complexity and (ii) automating refinement and solution execution. The technical complexity is hidden by the framework's declarative nature: management tasks are described only as abstract Management Annotations. All implementation details are inferred automatically by Management Planlets. Thus, the user only selects the idiom to be applied, the technical realization is invisible and automated. The second requirement of automation is achieved through executable transformations of idioms and the employed Management Planlet Framework.

D. Automatic Refinement Filtering and Pattern Configuration

To apply a pattern automatically using this system, the user first triggers the automatic ETG discovery of the application to be managed, searches the Semi-Automated Management Pattern (SAMP) to be applied, and selects the desired refinement in the form of an Automated Management Idiom. Based on their Topology Fragments, non-applicable Automated Management Idioms are filtered automatically by the system. For example, if the Stateless Component Swapping SAMP is refined by two different Automated Management Idioms—one is able to migrate a PHP application hosted on an Apache HTTP Server to Amazon EC2, the other is the described WAR migration idiom shown in Figure 10—the system offers only the idioms whose Topology Fragments match the elements in the ETG. Thus, in case of our motivating scenario, only the idiom for migrating the WAR application is offered as exclusively its Topology Fragment matches the ETG. Based on this matchmaking, the system is able to offer only applicable management patterns and refinements in the form of Automated Management Idioms. Therefore, we call this concept *Automatic Refinement Filtering*.

If multiple idioms of a selected SAMP match the ETG after filtering, e. g., one idiom migrates the WAR application to Amazon EC2, another to Microsoft's public Cloud "Azure", the user is able to *configure* the refinement of the pattern by a simple manual idiom selection while its actual application and execution are automated completely. Therefore, we call this concept *Configurable Automated Pattern Refinement*. The combination of these two concepts automates the refinement in two dimensions: first, Automatic Refinement Filtering preselects applicable refinements automatically, which relieves the user from finding applicable idioms. Second, the remaining filtered idioms can be applied directly without human intervention. Thus, the only manual step in this process is selecting the pattern and the desired configuration in the form of an idiom.

V. Validation and Evaluation

In this section, we validate the presented approach by a (i) prototypical implementation and evaluate the concept in terms of (ii) standards compliance, (iii) automation, (iv) technical complexity, (v) separation of concerns, and (vi) extensibility.

A. Prototype

To validate the technical feasibility of the approach, we extended the Java prototype presented in our former work [8] by Automated Management Idioms. The system's architecture is shown in Figure 11. The Web-based user interface is implemented in HTML5 using Java Servlets. Therefore, the prototype runs on a Tomcat 7 Servlet Container. The UI calls the *Application Management API* to apply a pattern to a running application. Therefore, it discovers the application's ETG by integrating the ETG Discovery Framework [18] and employs a *Pattern and Idiom Manager* to select the pattern / idiom to be applied. The manager is connected to a local library which stores deployable SAMPs and AMIs in the form of Java-based Webservices and implements a matchmaking facility to analyze which patterns / idioms are applicable to a certain ETG. To add new patterns or idioms to the system, the library provides an interface to register additional implementations. After selection, the discovered ETG and the SAMP / AMI to

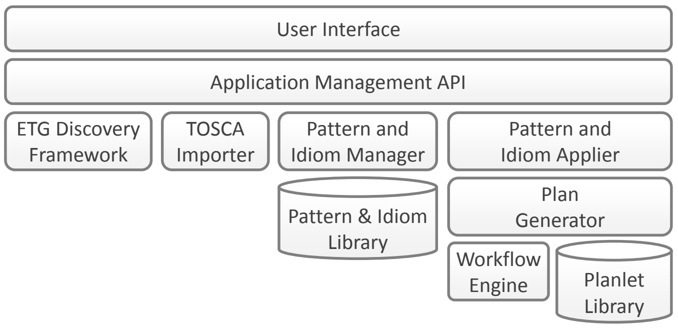

Figure 11. Fully Automated Pattern-based Management System Architecture.

be applied are passed to the *Pattern and Idiom Applier*, which executes the pattern's / idiom's transformation on the ETG and passes the resulting DASM to the *Plan Generator*. The generator is connected to a local *Management Planlet Library* that contains all available Planlets, which are implemented using the standardized Business Process Execution Language (BPEL) [25]. It generates the corresponding management workflow also in BPEL, which is deployed and executed afterwards on the employed workflow engine "WSO2 Business Process Server 2.0". We implemented several Semi-Automated Management Patterns and corresponding refinements in the form of Automated Management Idioms, e. g., for migration scenarios similar to the one used in this paper.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [26] is an OASIS standard to describe Cloud applications and their management in a portable way. Therefore, it provides a specification to model application topologies that can be linked with management workflows. As our approach is based on the same concepts, our prototype supports importing and deploying TOSCA-based applications, which can be managed afterwards using the presented approach.

B. Standards Compliance and Interoperability

Standards are a means to enable reusability, interoperability, and maintainability of software and hardware, which leads to higher productivity and helps aligning the enterprise's IT to its business. However, most available management approaches are based on non-standardized APIs or domain-specific languages (DSLs) which makes it difficult to provide and transfer the required knowledge. The presented approach tackles this issue by supporting two existing standards: (i) TOSCA is used to import standardized application descriptions that can be managed using SAMPs and AMIs while the (ii) BPEL standard is used to implement and execute the generated Management Workflows. In addition, Management Planlets and the generated workflows are implemented in BPEL, which is a standard to describe executable workflows. Thus, they are portable across standard-compliant BPEL engines. In addition, Planlets allow integrating different kinds of management technologies seamlessly [8][10]. As a result, the presented approach is agnostic to individual management technologies and Cloud providers, which supports interoperability and eases integration.

C. Automation

In Cloud Application Management, automation is of vital importance. Workflow technology enables automating the execution of management tasks and provides powerful features such as automated recoverability and compensation [9]. However, if these processes that implement a refined solution of the respective management pattern have to be created manually, it is not efficient and, in addition, error-prone. The presented approach automates the creation of workflows that implement a pattern's refined solution through the introduced Automated Management Idiom layer and the employed framework. Thus, only the choice when to trigger a management pattern is left to the human operator. This automation decreases the risk of human errors, which account for the largest fraction of failures and system downtimes in distributed systems [7][14]. In addition, the required technical knowledge to understand the different management technologies is not required at all as the whole process of determining the tasks to be performed and orchestrating the required technologies is fully automated.

D. Technical Complexity

Manually handling the technical complexity of pattern refinement for composite Cloud applications is a major issue due to heterogeneous and proprietary management technologies (cf. Section II). The presented approach tackles this issue by automating the whole refinement process from a pattern's abstract solution to the final executable workflow on two layers: (i) automating refinement by Automated Management Idioms and (ii) automated workflow generation using the Management Planlet Framework. The automated refinement removes the two manual tasks to specify concrete node and relationship types for abstract types and to add refinement-specific Management Annotations (cf. Figure 4 and Figure 7). Hence, the formerly required technical expertise on proprietary management idiosyncrasies such as Security Groups is no longer a mandatory prerequisite. Secondly, the orchestration of the different management technologies is completely handled by the Plan Generator that transforms the refined DASM fully automatically into an executable Management Workflow. Thus, this removes all manual steps and the only task that is left to the human user is choosing the AMI to be applied. This eliminates potential sources of errors on the technical layer [1].

E. Separation of Concerns

The presented approach separates concerns through splitting the process of selecting, creating, and automating a pattern, its refinement, and execution. Our approach enables IT experts to capture generic management knowledge in patterns that can be refined through Automated Management Idioms developed by specialists of certain areas. Thus, pattern creators and AMI developers are separate roles that are responsible for different kinds of knowledge: SAMP creators capture *generic* management knowledge, AMI creators refine this through implementing concrete *technology-specific* management knowledge. In addition, experts of low-level API orchestration that understand the different management technologies are able to automate their knowledge through implementing Management Planlets. Thus, even AMI creators do not have to deal with complex technology-specific details such as API invocations or parametrization: they only implement their knowledge declaratively, i. e., without defining the final API calls etc. This enables separating different responsibilities and concerns as well as a seamless integration of different kinds of knowledge.

F. Extensibility

The presented approach is extensible on multiple layers as it provides an explicit integration framework for patterns, idioms, and Planlets through using libraries. New Semi-Automated Management Patterns and Automated Management Idioms can be created based on a uniform Java interface and integrated into the system seamlessly by a simple registration at the library. All patterns and idioms in the library are considered automatically when an application shall be managed by comparing their Topology Fragments with the application's ETG. To extend the system in terms of management technologies, Planlets can be implemented for new node types, relationship types, or Management Annotations and stored in the Planlet Library. The framework's Plan Generator integrates new Management Planlets without further manual effort when processing DASMs.

VI. Related Work

Several works focus on automating application management in terms of application provisioning and deployment. Eilam et al. [27], Arnold et al. [28], and Lu et al. [29] consider pattern-based approaches to automate the deployment of applications. However, their *model-based patterns* are completely different from the abstract kind of patterns we consider in this paper. In their works, patterns are topology models which are used to associate or derive the corresponding logic required to deploy the combination of nodes and relations described by the topology, similarly to the Annotated Topology Fragments of Management Planlets. Mietzner [30] presents an approach for generating provisioning workflows by orchestrating "Component Flows", which implement a uniform interface to provision a certain component. However, these works focus only on provisioning and deployment and do not support management.

Fehling et al. [1][15] present *Cloud Application Management Patterns* that consider typical management problems, e. g., the "Stateless Component Swapping Pattern", which was automated in this paper. They propose to attach abstract processes to management patterns that describe the high-level steps of the solution. However, these abstract processes are not executable until they are refined manually for individual use cases. Their management patterns also define requirements in the form of architectural patterns that must be implemented by the application. These dependencies result in a uniform pattern catalog that interrelates abstract management patterns with architectural patterns. In addition, they propose to annotate reusable "Implementation Artifacts" to patterns that may assist applying the pattern, e. g., software artifacts or management processes. Their work is conceptually equal to our approach in terms of using processes to automate pattern application. However, most of the refinement must be done manually, which leads to the drawbacks discussed in Section I: management processes must be created in advance to be executable when needed. In addition, changing application structures caused by other pattern applications, for example, migration patterns, lead to outdated processes that possibly result in reimplementation. In Falkenthal et al. [31], we show how reusable solution implementations, e. g., Management Workflows, can be linked with patterns. However, also this approach requires at least one manual implementation of the concrete solution which is typically tightly coupled to a certain application structure.

Fehling et al. [32] also present a step-by-step pattern identification process supported by a pattern authoring toolkit. This authoring toolkit can be combined with our approach to create patterns and idioms that can be automated afterwards. Thus, the work of Fehling et al. provides the basis for creating AMIs out of patterns captured in natural text following this process.

Reiners et al. [33] present an iterative pattern evolution process for developing patterns. They aim for documenting and developing application design knowledge from the very beginning and continuously developing findings further. Non-validated ideas are documented already in an early stage and have to pass different phases until they become approved design patterns. This process can be transferred to the domain of Cloud Application Management Patterns. Our approach supports this iterative pattern evolution process as it helps to apply captured knowledge easily and quickly to new use cases. Thus, it provides a complementary framework to our approach that enables validating and capturing knowledge.

VII. Conclusion and Future Work

In this paper, we presented the concept of Automated Management Idioms that enables automating the refinement and execution of a pattern's abstract solution automatically to a certain use case by generating executable management workflows. We showed that the approach enables (i) applying the concept of patterns efficiently in the domain of Cloud Application Management through automation, (ii) abstracting the technical complexity of refinement, and (iii) reducing human intervention. The approach enables operators to apply various management patterns fully automatically to individual use cases without the need for detailed technical expertise. The prototypical validation, which extends the Management Planlet Framework, proves the concept's technical feasibility. In future work, we plan to investigate how Automated Management Idioms can be triggered automatically based on occurring events and how multiple patterns can be applied together.

Acknowledgment

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

References

[1] C. Fehling, F. Leymann, J. Rütschlin, and D. Schumm, "Pattern-based development and management of cloud applications." Future Internet, vol. 4, no. 1, March 2012, pp. 110–141.

[2] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2003.

[3] A. Nowak et al., "Pattern-driven Green Adaptation of Process-based Applications and their Runtime Infrastructure," Computing, February 2012, pp. 463–487.

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

[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep., July 2009.

[6] M. Armbrust et al., "A view of cloud computing," Communications of the ACM, vol. 53, April 2010, pp. 50–58.

[7] A. B. Brown and D. A. Patterson, "To err is human," in EASY, July 2001, p. 5.

[8] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Pattern-based runtime management of composite cloud applications," in CLOSER. SciTePress, May 2013, pp. 475–482.

[9] F. Leymann and D. Roller, Production workflow: concepts and techniques. Prentice Hall PTR, 2000.

[10] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and J. Wettinger, "Integrated cloud application provisioning: Interconnecting service-centric and script-centric management technologies," in CoopIS. Springer, September 2013, pp. 130–148.

[11] S. Leonhardt, "A generic artifact-driven approach for provisioning, configuring, and managing infrastructure resources in the cloud," Diploma thesis, University of Stuttgart, Germany, November 2013.

[12] F. Leymann, "Cloud Computing: The Next Revolution in IT," in Proc. 52th Photogrammetric Week, September 2009, pp. 3–12.

[13] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, January 2014.

[14] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in USITS. USENIX Association, June 2003, pp. 1–16.

[15] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas, "Service Migration Patterns - Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments," in SOCA. IEEE, December 2013, pp. 9–16.

[16] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group Std., Rev. 2.0, January 2011.

[17] T. Binz, C. Fehling, F. Leymann, A. Nowak, and D. Schumm, "Formalizing the Cloud through Enterprise Topology Graphs," in CLOUD. IEEE, June 2012, pp. 742–749.

[18] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Automated Discovery and Maintenance of Enterprise Topology Graphs," in SOCA. IEEE, December 2013, pp. 126–134.

[19] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and D. Schumm, "Vino4TOSCA: A visual notation for application topologies based on TOSCA," in CoopIS. Springer, September 2012, pp. 416–424.

[20] Apache Software Foundation. Apache Tomcat 7 API. [Online]. Available: http://tomcat.apache.org/tomcat-7.0-doc/

[21] Amazon Web Services. Elastic Compute Cloud API Reference. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/APIReference

[22] United Domains. Reselling API. [Online]. Available: http://www.ud-reselling.com/api

[23] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and M. Wieland, "Policy-Aware Provisioning of Cloud Applications," in SECURWARE. Xpert Publishing Services, August 2013, pp. 86–95.

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

[25] OASIS, Web Services Business Process Execution Language (WS-BPEL) Version 2.0, OASIS, April 2007.

[26] OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0, May 2013.

[27] T. Eilam, M. Elder, A. Konstantinou, and E. Snible, "Pattern-based composite application deployment," in IM. IEEE, May 2011, pp. 217–224.

[28] W. Arnold, T. Eilam, M. Kalantar, A. V. Konstantinou, and A. A. Totok, "Pattern based soa deployment," in ICSOC. Springer, September 2007, pp. 1–12.

[29] H. Lu, M. Shtern, B. Simmons, M. Smit, and M. Litoiu, "Pattern-based deployment service for next generation clouds," in SERVICES. IEEE, June 2013, pp. 464–471.

[30] R. Mietzner, "A method and implementation to define and provision variable composite applications, and its usage in cloud computing," Dissertation, University of Stuttgart, Germany, August 2010.

[31] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, "From Pattern Languages to Solution Implementations," in PATTERNS. Xpert Publishing Services, May 2014.

[32] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rütschlin, and D. Schumm, "Capturing cloud computing knowledge and experience in patterns," in CLOUD. IEEE, June 2012, pp. 726–733.

[33] R. Reiners, "A pattern evolution process - from ideas to patterns." in Informatiktage. GI, September 2012, pp. 115–118.