Automatic Topology Completion of TOSCA-based Cloud Applications

Pascal Hirmer[1], Uwe Breitenbücher[2], Tobias Binz[2], Frank Leymann[2]

Pascal.Hirmer@ipvs.uni-stuttgart.de
[1]Institute for Parallel and Distributed Systems
[2]Institute of Architecture of Application Systems
University of Stuttgart

Abstract: Automation of application provisioning is one of the key aspects of Cloud Computing. Standards such as the Topology and Orchestration Specification for Cloud Applications (TOSCA) provide a means to model application topologies which can be provisioned fully automatically. Many automated provisioning engines require that these topologies are complete in the sense of specifying all application, platform, and infrastructure components. However, modeling complete models is a complex, time-consuming, and error-prone task that typically requires a lot of technical expertise. In this paper, we present an approach that enables users to model incomplete TOSCA application topologies that are completed automatically to deployable, complete models. This enables users to focus on the business-relevant application components and simplifies the creation process tremendously by minimizing the required effort and know-how. We prove the technical feasibility of the presented approach by a prototypical implementation based on the open source modeling tool Winery. In addition, we evaluate the approach by standards-compliance and performance.

1 Introduction and Background

In recent years, Cloud Computing gained a lot of attention due to its economical benefits through payment on demand, flexibility, scalability and self-service [MG11]. One key to enable this is full automation of application provisioning and management. To make this possible, standards have been developed to automate the provisioning process. One example is the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [OA13a], a standard that enables creating portable application models including management functionalities [Bi14]. Such topology models are called *Topology Templates* and describe the application's components and their relationships as directed topology graph. Topology Templates are usually modeled by *Cloud Service Developers* with the aim to deploy their developed applications in a Cloud environment. However, Cloud Service Developers – as defined in [OA13b] – are no experts in Cloud infrastructure and platform components. For non-experts, modeling a Topology Template for applications consisting of multiple different components that run on heterogeneous Cloud offerings is a complex task: (i) the configuration of components must be defined, (ii) interoperability between systems has to be considered, and (iii) the artifacts for deployment must be attached to enable a fully automated provisioning of the application. Thus, a lot of technical know-how regarding Cloud infrastructures, platforms, providers and software components is required. As a result, modeling complex applications leads to huge effort causing high costs.

The result of this discussion is that it is of vital importance to ease the modeling of complex Cloud applications to reduce the required know-how. As we tackle these issues, the driving research question of this paper is: *How can the (i) manual effort and (ii) technical expertise currently required for modeling TOSCA topology models be minimized?* We answer this question by presenting an approach that enables modelers to create *incomplete topology models* that are completed automatically by the system we present. Note that in the context of this paper, the term *incomplete* is used as a synonym for the term *undeployable*. An *undeployable* topology can't be used for automatic provisioning due to missing components. In these incomplete models, only the business-relevant components have to be specified by the modeler—the required underlying infrastructure and platform components including their relations are inserted fully automatically while guaranteeing a syntactically and semantically correct model. The resulting models can be used as input for automated TOSCA provisioning engines that require complete models. The presented approach tackles the mentioned issues in three different dimensions: first, the complexity of modeling is reduced as only business-relevant information has to be provided. Second, the required manual effort for modeling is reduced. Third, as the approach is based on TOSCA, all models are portable which helps to avoid vendor lock-in. All these properties help to decrease the costs of developing, provisioning, and maintaining Cloud-based applications, thus, leading to economical advantages when modeling applications and their management using TOSCA. We prove the technical feasibility of the approach by a prototypical implementation and evaluate the concept in terms of standards-compliance and performance.

The remainder of this paper is structured as follows: after presenting related work in Section 2, we introduce the basics of TOSCA that are required to understand the presented approach in Section 3. In the following Section 4, we provide a motivating scenario which is used throughout the paper to explain the developed concepts. In Section 5, the main contribution of this paper is presented: we show a concept to automatically complete incomplete TOSCA topologies. We evaluate in Section 6, validate the approach in Section 7, and give a conclusion as well as an outlook on future work in Section 8.

2 Related Work

This section discusses related works which have been published in this research field and serve as basis for the presented approach. The article "Managing the Configuration Complexity of Distributed Applications in Internet Data Centers" [Ei06] presents an approach for a model driven provisioning and management of applications for the TOSCA standard. Using transformations, topologies are transformed to several levels of abstraction. Similar to the approach presented in this paper, the resulting topology can be used for provisioning using a provisioning engine. Arnold et al. [Ar07] present a pattern based approach for the deployment of service-oriented architectures. By defining patterns for certain deployment scenarios (e.g. high availability), the deployment can be managed dependent on the requirements. These pattern are very similar to the concept of Requirements and Capabilities in the TOSCA standard. Binz et al. [Bi12] present an approach to use topologies to describe complex enterprise IT systems. These topologies are called *Enterprise Topology Graphs (ETG)* and describe the system components and their connections similar to TOSCA. However, while TOSCA defines topologies for the provisioning of applications, ETGs describe the state of already provisioned systems. Breitenbücher et al. [Br12] present Vino4TOSCA, a visual notation of the TOSCA standard. In this paper we use the Vino4TOSCA notation to present our approach graphically. Vino4TOSCA is also used for the implementation of the modeling tool Winery [Ko13]. Eilam et al. [Ei11] and Kalantar et al. [Ei06] present a combination of a model and workflow driven approach for the deployment of applications. A deployment model describing the final state is used to create a workflow model containing operations to transform an initial topology to the final state. Using a workflow engine this transformation can be processed automatically. Képes [Ke13] presents an implementation of the Provisioning API for OpenTOSCA (cf. Section 5.3.1), which can be used for the automatic completion of topologies that shall be provisioned in the OpenTOSCA Runtime Environment. Finally, Brogi et al. [BS13] present an approach to match Service Templates and Node Types, also using the concepts of TOSCA Requirements and Capabilities.

3 TOSCA

In this section, we introduce the basic concepts of TOSCA, which are relevant to understand the approach presented in this paper. TOSCA is an OASIS standard introduced in November 2013 to describe Cloud applications in a portable way. TOSCA-based descriptions define (i) the structure as well as (ii) management functionalities of Cloud-based applications, e.g., management functionalities can be implemented using executable management plans. Although TOSCA is a relatively new standard, several tools exist that ease modeling, provisioning, and management of TOSCA-based applications. The open source ecosystem *OpenTOSCA*, for example, includes a graphical modeling tool called *Winery* [Ko13] and a plan-based provisioning and management Runtime Environment [Bi13] which can be used to provision and manage TOSCA applications fully automatically by executing management plans. The TOSCA Primer [OA13b] defines the roles *Cloud Service Consumer*, *Cloud Service Developer* and *Cloud Service Provider*. The approach presented in this paper mainly supports the *Cloud Service Developer*, who models TOSCA topologies to provision and manage an application in the Cloud. Further details on the TOSCA standard can be found in the official OASIS TOSCA specification [OA13a], TOSCA Primer [OA13b], or Binz et al. [Bi14].

The core of the application description in TOSCA is the *Topology Template*, a directed graph containing *Node Templates* (vertices) and *Relationship Templates* (edges). Node Templates may describe all components of an application, including all software and hardware components. The relations between those Node Templates are represented by Relationship Templates. Node and Relationship Templates are typed by *Node Types* and *Relationship Types*, respectively. Types define the semantics of the templates, as well as their properties, provided management operations, and so on. Types can be refined or extended by an inheritance mechanism. *TOSCA Requirements* are used to define requirements and restrictions for a Topology Template. Based on its Requirements, it can be determined if a Node Template must be connected and, as a consequence, if further Node Templates are missing in the Topology Template to satisfy all Requirements. Each Requirement is derived from a *Requirement Type* defining the structure and semantics of the Requirement, which makes it possible to process Requirements based on their QName. To *fulfill* the Requirements of a Topology Template, every Node Template containing a TOSCA Requirement has to be connected to a Node Template containing a suitable *TOSCA Capability* using a Relationship Template. Which type of Capability is demanded by the Requirement is described by the XML attribute *requiredCapabilityType* of its Requirement Type. Besides being attached to Node Templates, Requirements can also be attached to Node Types as Requirement Definitions which makes them Requirements for all Node Templates of this Node Type. TOSCA specifies an exchange format called *Cloud Service Archive* (CSAR) to package Topology Templates, types, associated artifacts, plans, and all required files into one self-contained package. This package is portable across different standards-compliant TOSCA Runtime Environments [Br14a].

4 Motivating Scenario

This section introduces the motivating scenario that is used throughout the paper to illustrate our approach. It describes the modeling and completion of a TOSCA Topology Template to provision a Web-application on Amazon EC2[1]. The application used in this scenario consists of a Java-based frontend and a MySQL database to store its data. This application shall be hosted in the Amazon Cloud to be accessible remotely. As described in Section 1, in some TOSCA runtimes (e.g. OpenTOSCA), a complete and technically detailed TOSCA Topology Template has to be modeled to enable automatic provisioning. To simplify the modeling of such – currently XML-based – topologies, the Cloud Service Developer may use the graphical TOSCA modeling tool Winery [Ko13]. However, even with graphical modeling, the application programmer requires extensive knowledge of the technical details for provisioning Cloud applications as the whole topology model has to be specified in detail. The approach presented in this paper enables also inexperienced users to provision applications in the Cloud using TOSCA. This is enabled by allowing users to model incomplete topologies that do not specify the whole topology in detail but only the business-relevant components. An incomplete model for this motivation scenario is shown in Figure 1 on the left: the shown Topology Template is incomplete as it only contains two Node Templates, one representing the Java-based Web application packed as Java Web Archive (WAR), and the MySQL database, without defining the underlying infrastructure etc. However, this topology cannot be provisioned by runtimes that require complete models (e.g., OpenTOSCA): (i) the Webserver for running the Java frontend and

[1]https://aws.amazon.com/de/ec2/

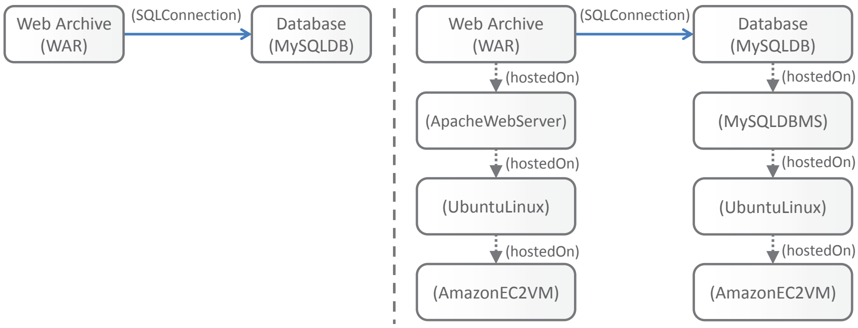


Figure 1: Incomplete (left) and complete (right) Topology Templates of the Motivation Scenario

database management system components are missing, (ii) the operating systems are not specified, and (iii) no information is provided about where the stacks have to be deployed. Thus, to make an automatic provisioning of this application possible, the missing technical information about the infrastructure and middleware components have to be defined. An example of a complete topology model that may result from applying our approach is shown in Figure 1 on the right: this model contains all middleware and infrastructure components and provides, therefore, a complete model that can be used to provision the application (we omitted properties, operations etc. to simplify the figure). In the next section, we present our approach for topology completion that supports the completion of incomplete Topology Templates.

5 Automated Completion of Incomplete TOSCA Topology Templates

In this section, we present the main contribution of this paper in the form of an approach for automatic TOSCA Topology Template completion. We describe the presented approach as method that is subdivided into five steps, which are shown in Figure 2 and explained in detail in the following subsections: (i) modeling of incomplete Topology Templates, (ii) target runtime selection, (iii) automated completion to deployable model, (iv) optional manual refinement of the completed solution, and (v) final deployment of the application.

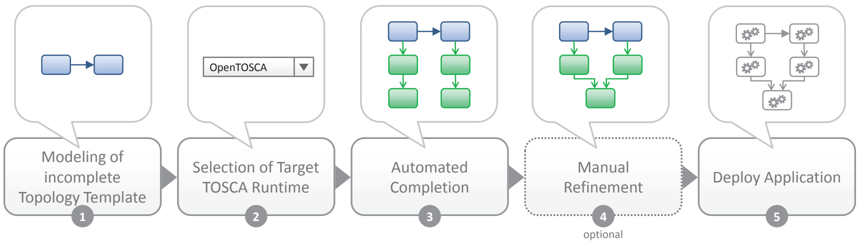


Figure 2: Automated TOSCA Topology Template Completion Method

5.1 Step 1: Modeling of Incomplete Topologies using TOSCA Requirements

In the first step, an incomplete Topology Template defining Requirements (cf. Section 3) as hints for the automatic completion is modeled. We define an incomplete topology as *a Topology Template that cannot be used for automated provisioning due to missing components*. Thus, considering the motivating scenario, users do not want to care about details such as the employed Cloud provider or the used operating system. However, there are two general requirements that have to be fulfilled: the application has to be deployed on a Web server and the MySQL database on a compatible database management system (DBMS). Consequently, these components must be added automatically by the topology completion based on the Requirements. To enable this, TOSCA Requirements have to be declared in the incomplete topology model. Therefore, the Requirements are either (i) defined directly by the Requirement Definitions of the corresponding Node Types (which is the common case) or (ii) added manually by the Cloud Service Developer (e.g., to additionally refine Requirements). In our example, two Requirements "WebserverContainerRequirement" and "MySQLDBMSRequirement", as shown in Figure 3, are defined. The result is an incomplete Topology Template that contains all application-specific components including a description of the corresponding Requirements that must be fulfilled to run the application.

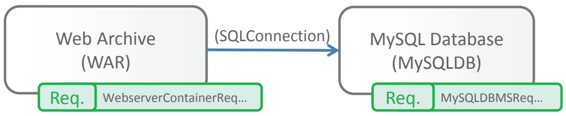


Figure 3: Incomplete TOSCA Topology Template with Requirements

5.2 Step 2: Selection of Target TOSCA Runtime Environment

The TOSCA standard specifies a meta-model for describing Cloud application structures including all components and relationships among them. However, the standard currently neither predefines Node Types nor Relationship Types. Thus, there are *no* well-defined semantics of Node and Relationship Types. As a consequence, different TOSCA Runtime Environments (which are not standardized) typically support different features, containers, and Cloud providers. For example, the OpenTOSCA Runtime Environment [Bi13] currently supports automatically deploying applications on Amazon EC2 but not on Microsoft Azure. Thus, it supports the deployment based on the *AmazonEC2* Node Type but not on the *Azure* Node Type. Therefore, the completion of incomplete Topology Templates must consider the final TOSCA Runtime Environment and the supported features. To be precise, the completion must be aware which Node and Relationship Types can be provisioned by the employed runtime. Therefore, we have to select the target environment in this step to provide the required information for completing the model appropriately.

5.3 Step 3: Automatic Completion of Incomplete TOSCA Topologies

TOSCA topology completion can be done in two ways, the Black Box and the Glass Box approach: In the *Black Box approach*, used for example by inexperienced users, there is no user interaction during the completion of the topology, i.e., all decisions are made automatically. An incomplete TOSCA Topology Template (cf. Section 5.1) serves as input and one or more completed topologies are returned. If there is more than one resulting topology, the user can choose after the completion finished. In contrast, the *Glass Box approach* enables the user to control and observe the topology completion step by step. Therefore, if multiple possibilities are found to fulfill one Requirement or to connect two Node Templates, the user will be prompted to decide interactively. Using this approach is reasonable if the user has a concrete vision of the resulting completed topology.

5.3.1 Basics

Before presenting the topology completing algorithm in Section 5.3.2, this section introduces concepts on how to connect two Node Templates, how to check a topology for being provisionable in a TOSCA Runtime Environment, and introduces a Type Repository.

Connecting Node Templates: During the topology completion, Node Templates will be added to the topology. These have to be connected to the existent topology using Relationship Templates. To help determining the right Relationship Type to connect two Node Templates, the TOSCA standard allows to define valid source and target Node Types in each Relationship Type. These elements contain a reference to the Node Types that can be connected using the Relationship Type. Furthermore, these elements can also contain Requirement and Capability Types. To find a Relationship Type that is able to connect two Node Templates, all available Relationship Types are checked. If the type of these Node Templates is contained in the ValidSource and ValidTarget elements of a Relationship Type it can be used. In case the Node Templates contain Requirements or Capabilities, it is checked by their types if the Relationship Type is able to connect them. After the algorithm finished, the Relationship Type is instantiated to a Relationship Template – containing references to the connected Node Templates – and inserted into the topology.

Provisioning API: The completion algorithm requires information about the supported features of the target TOSCA Runtime Environment which shall provision the topology. This information is made available through the *Provisioning API*, which must be provided by each TOSCA Runtime Environment that shall be supported by our approach. The following three operations are implemented by the Provisioning API: (i) Check if a given Topology or Node Template can be automatically provisioned by the respective TOSCA Runtime Environment, (ii) propose Node and Relationship Templates to be inserted below a given Node Template that shall be provisioned, and (iii) check if the TOSCA Runtime Environment is able to handle a certain TOSCA Requirement directly without inserting additional nodes. In case a Requirement can be handled this way, it doesn't have to be fulfilled by the topology completion algorithm.

Type Repository: To complete a Topology Template, Node and Relationship Templates are inserted based on predefined Node and Relationship Types, which are made available to the completion algorithm through a type repository.

5.3.2 Topology Completion – Algorithm

In this section, we present the algorithms used to complete incomplete TOSCA Topology Templates to be provisioned on a given target TOSCA Runtime Environment. In addition, the flag *FulfillAllRequirements* indicates whether all Requirements have to be fulfilled or just the Requirements that cannot be handled by the TOSCA Runtime Environment. The completion of a topology is done in two steps: (i) fulfillment of Requirements and (ii) verification of completeness. To simplify the presentation, the Black Box approach is discussed here, i.e., proceeding without any user interaction. In the Glass Box approach, in case there are multiple possibilities to process, the user is prompted to choose one.

Step i: Fulfillment of Requirements: First, the Requirement analysis iterates over all the Node Templates of a Topology Template and their Types to check for existing Requirements or Requirement Definitions. In case a Requirement exists at a Node Template or Type, it is checked for fulfillment: If the Node Template containing a Requirement is connected to a Node Template containing a suitable Capability, the Requirement is already fulfilled. If required (flag *FulfillAllRequirements* is set to *false*), it is also checked whether the TOSCA Runtime Environment can already handle the Requirement using the Provisioning API introduced in Section 5.3.1. If that's the case, the Requirement can also be marked as fulfilled. For non-fulfilled Requirements, the required Capability Type is found by checking the *requiredCapabilityType* attribute of its Requirement Type. Thereupon, a Node Type is searched in the repository offering the suitable Capability Type. In case a suitable type can be found, a Node Template is inserted into the Topology Template by connecting it to the Node Template containing the Requirement (cf. Section 5.3.1). In this way all Requirements are processed and fulfilled. Finally, the analysis phase is restarted to check whether new Requirements have been added through the types of the inserted Node Templates. This leads to a recursive algorithm finishing when all Requirements have been fulfilled.

Step ii: Verification of Completeness: A topology without non-fulfilled Requirements does not guarantee that it can be provisioned in the selected TOSCA Runtime Environment. This is the case if the Runtime Environment requires further components in the topology for provisioning. Those components possibly were not added by the topology completion due to missing TOSCA Requirements or Node Types in the Type Repository. Therefore, using the Provisioning API introduced in Section 5.3.1 it is checked whether the completed topology can be automatically provisioned in the used TOSCA Runtime Environment or not. If the topology cannot be provisioned, the API returns a Topology Template containing the missing Templates which will be inserted into the topology. In case any Node Templates are added in Step ii, the first step is re-processed because new Requirements could have been added through the types of the inserted Node Templates.

Result: After the algorithm has finished, the topology is complete and can be used for provisioning in the selected TOSCA Runtime Environment.

5.4 Step 4: Manual Refinement

After the execution of the completion algorithm, the user can refine the resulting topology manually. For example, the user might want to switch the Cloud provider, which can be done by changing the type of the VM Node Templates. Note that it is necessary to check the provisionability of the refined topology again after changing it manually.

5.5 Step 5: Deploy Application

In the last step of the method, the complete application topology gets deployed on the selected TOSCA Runtime Environment. In Breitenbücher et al. [Br14a], we show how Topology Templates are interpreted and provisioned fully automatically using OpenTOSCA.

6 Evaluation

In this section, we conduct a qualitative and quantitative evaluation based on a toolchain, standards compliance, as well as the completion's performance.

Toolchain—End-to-End Prototype Support: The presented approach bridges the gap between the modeling of TOSCA topologies and the provisioning of applications. We developed the open source tool Winery that provides an user interface to model TOSCA topologies graphically and offers a management backend to manage types and artifacts. Types and topologies can be exported as CSAR files. Furthermore, a TOSCA Runtime Environment has been developed called OpenTOSCA [Bi13] which is able to run those CSARs. Using the plan generator presented in [Ke13] and [Br14a], provisioning plans can be generated and injected fully automatically into the CSAR running in the OpenTOSCA Runtime Environment. The last link of the toolchain is the "Vinothek" [Br14b] providing a graphical end-user interface to provision new application instances using OpenTOSCA. With our implementation of the presented approach, Topology Templates can be completed and provisioned fully automatically using the OpenTOSCA ecosystem. In conclusion, we provide tools to support an end-to-end TOSCA toolchain, as shown in Figure 4.

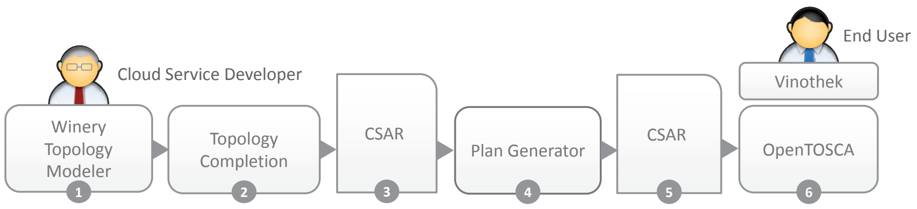


Figure 4: End-to-end Tool support

Standards Compliance: Standards are a means to achieve reusability, interoperability, portability, and maintainability resulting in higher productivity and lower cost. The approach presented in this paper addresses these issues as it exclusively employs the OASIS standard TOSCA as exchange format and is integrated into a chain of further TOSCA-compliant tools. Thus, the approach enables portability [Bi14] and automatic provisioning of applications based on standards.

Performance Measurement: Several performance measurements have been conducted, showing the runtime of the topology completion algorithm. The measurements are based on timestamps taken of the prototypical implementation. As displayed in Table 1, the completion algorithm has an exponentially growing run time depending on the amount of Requirements being completed. The measured runtime can be explained with the recursive algorithm being used.

Table 1: Topology Completion Duration

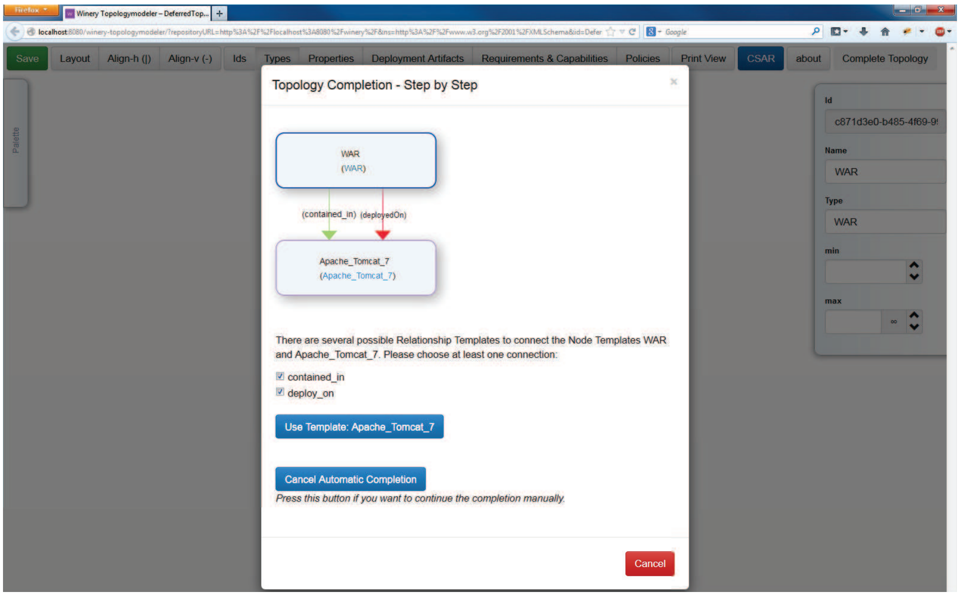
# Requirements	Completion Time ∅	Time / Requirement ∅
50	31 ms	0.62 ms
250	47 ms	0.18 ms
500	63 ms	0.12 ms
2 500	296 ms	0.11 ms
5 000	1 014 ms	0.20 ms
25 000	34 897 ms	1.39 ms
50000	168600 ms	3.372 ms

7 Validation & Prototype

The validation is based on the prototypical implementation we developed and integrated into the TOSCA modeling tool *Eclipse Winery* [Ko13]. Winery is an Eclipse open source project[2] and the Winery Wiki provides further information on the topology completion facilities we integrated[3]. Using Winery, the user graphically models incomplete topologies using the Vino4TOSCA visual notation [Br12]. The completion process is started from Winery by clicking the button "Complete Topology" and selecting the approach to use, as well as the targeted TOSCA Runtime Environment. In case the Glass Box approach is chosen, in every step, the completion gives feedback about the Node Templates that could be inserted into the topology and prompts the user to select the desired one (cf. Figure 5). When using the Black Box approach, the topology will be completed without further user interaction. The completed topology is then displayed in Winery.

[2]http://eclipse.org/winery/

[3]https://wiki.eclipse.org/Winery/Addons/Winery_Topology_Completion



There are several possible Relationship Templates to connect the Node Templates WAR and Apache_Tomcat_7. Please choose at least one connection:

☑ contained_in
☑ deploy_on

Press this button if you want to continue the completion manually.

Id	c871d3e0-b485-4f69-9
Name	WAR
Type	WAR
min	
max	∞

Figure 5: Topology Completion in Winery

8 Conclusion and Future Work

In this paper, we presented an approach to simplify the modeling of TOSCA topologies by automatically completing incomplete topologies. The completed topologies can then be automatically provisioned in the respective TOSCA Runtime Environment. We enable the user to create incomplete topologies that only define application components without giving information about platforms and infrastructure. These topologies are completed by adding middleware and infrastructure components until the topology is complete and can be processed by a provisioning engine. The user can control the completion using the presented Glass Box approach enabling him to determine which components will be added to the topology. The completion can also be processed without any user interaction by using the Black Box approach to reduce the modeling effort to a minimum. As mentioned before, the presented solution is usually used by Cloud Service Developers, exclusively. These developers want to benefit from the advantages of the TOSCA standard to enable automatic provisioning and management of their applications in the Cloud, without caring about infrastructure and platform details. Using the presented approach, it is sufficient to define information about the application to be deployed and ignore all other details. In practical use, this leads to an easy, low-effort way to deploy applications in the Cloud while also profiting from the advantages of the TOSCA standard. In conclusion, the presented solution is highly applicable for practical scenarios. In the future, the presented concepts can be improved by including the completion of TOSCA properties (e.g. IP addresses of servers, user / passwords of database etc.). In this approach the TOSCA properties are completed only with predetermined default values. Furthermore, the current approach can be extended in the future by using TOSCA Policies to define non-functional requirements (e.g. security, costs, availability) to steer the topology completion.

Acknowledgment

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

References

[Ar07] Arnold et al. Pattern Based SOA Deployment. In *ICSOC*, pages 1–12. Springer, 2007.

[Bi12] Tobias Binz et al. Formalizing the Cloud through Enterprise Topology Graphs. In *CLOUD*, pages 742–749. IEEE, June 2012.

[Bi13] Tobias Binz et al. OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *ICSOC*, pages 692–695. Springer, December 2013.

[Bi14] Tobias Binz et al. *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, pages 527–549. Advanced Web Services. Springer, January 2014.

[Br12] Breitenbücher et al. Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In *CoopIS*, pages 416–424. Springer, September 2012.

[Br14a] Breitenbücher et al. Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *IC2E*, pages 87–96. IEEE, March 2014.

[Br14b] Breitenbücher et al. Vinothek - A Self-Service Portal for TOSCA. In *ZEUS*, pages 69–72. CEUR-WS.org, March 2014.

[BS13] Brogi and Soldani. Matching Cloud Services with TOSCA. In *Advances in Service-Oriented and Cloud Computing*, pages 218–232. Springer, 2013.

[Ei06] T. Eilam et al. Managing the configuration complexity of distributed applications in Internet data centers. *Communications Magazine, IEEE*, 44(3):166–177, March 2006.

[Ei11] T. Eilam et al. Pattern-based Composite Application Deployment. In *IM*, pages 217–224. IEEE, May 2011.

[Ke13] Kalman Kepes. Konzept und Implementierung eine Java-Komponente zur Generierung von WS-BPEL 2.0 BuildPlans für OpenTOSCA. Bachelor Thesis: University of Stuttgart, Institute of Architecture of Application Systems, July 2013.

[Ko13] Oliver Kopp et al. Winery – A Modeling Tool for TOSCA-based Cloud Applications. In *ICSOC*, pages 700–704. Springer, December 2013.

[MG11] Mell and Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), September 2011.

[OA13a] OASIS. Topology and Orchestration Specification for Cloud Applications. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html, Nov 2013.

[OA13b] OASIS. TOSCA Primer. http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf, November 2013.