Towards Software Compliance Specification and Enforcement Using TOSCA

Mohammed Mubarkoot(✉) and Jörn Altmann(✉)

Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul 08826, Korea
mubarkoot@snu.ac.kr, jorn.altmann@acm.org

Abstract. According to the laws of software evolution, the size and complexity of software systems continue to increase over time and, simultaneously, if not maintained rigorously, the quality decreases. Quality degradation typically happens due to changes in policies, regulations, and industry requirements, which, in turn, complicates compliance management over time. Among the key challenges in managing the evolution of software are the modelling and the enforcement of compliance rules. Moreover, the gap between compliance experts and software engineers has worsened the problem. The topology and orchestration specifications for cloud applications (TOSCA), which is an OASIS standard, has the potential to offer a relief by enabling different levels of abstractions for modeling and enforcing compliance policies. This work aims at investigating the potential of using TOSCA service templates for modelling and enforcing non-functional requirements and policies. Then, it proposes an approach that maximizes involvement of stakeholders in modeling and auditing such requirements and policies. Findings can help enterprises and policy makers achieve better governance and compliance on software services.

Keywords: Software compliance · Non-functional requirements · Software evolution · Stakeholders' involvement · TOSCA blueprint

1 Introduction

Compliance management is one of the critical challenges in all stages of the software development life cycle (SDLC). In particular, the E-type software evolves over time as a response to real world changes. This continuous change increases the complexity and, as a result, leads to a degradation in quality if not maintained well [1]. In addition to that, the continuous changes of policies and industry-specific requirements further complicates governance and compliance management of a software. Lehman's laws of software evolution, namely continuing change and growth, increasing complexity, declining quality and feedback system, still apply and cannot be ignored [1]. Therefore, such continuous changes make it difficult to track whether the overall changes made in the software adhere to corporate policies and compliance requirements; and more importantly, getting insights on the status of policy modeling and enforcement at different levels of abstraction for different stakeholders.

K. Tserpes et al. (Eds.): GECON 2021, LNCS 13072, pp. 168–177, 2021.
https://doi.org/10.1007/978-3-030-92916-9_14

The recent decades experienced a huge change in the software industry in areas of distributing development, crowdsourcing, service-oriented approaches, and microservice practices [23]. This change is also powered by a big shift to cloud computing, which leads to more standardized software services [2]. The laws that govern software evolution do not seem to have adapted to the new paradigm shifts [1].

In this regard, many cloud modeling languages were introduced to address issues related to modeling and specification of cloud applications. Bergmayr et al. [1] conducted a systematic review on existing cloud modeling languages. They found that the majority of the existing modeling languages focus primarily on design-time aspects and very few consider the provisioning and runtime aspects. The topology and orchestration specifications for cloud applications (TOSCA) can contribute to the convergence of different cloud modeling languages, besides its abilities to describe processes for creating, terminating cloud services and for managing them throughout their whole lifetime [3]. According to the Organization for the Advancement of Structured Information Standards (OASIS) [3], TOSCA provides strong typing for artifacts in addition to the ability to extend to new types without extending the language definition [4]. Compared to other modeling languages, TOSCA supports the decomposition of software and definition of policies and non-functional behavior of a system [4]. It also implements management plans using existing workflow languages, namely the business process model and notation (BPMN) and the business process execution language (BPEL) [5]. This makes it promising for modeling non-functional requirements and enhancing evolution management of a software.

The aim of this paper is to explore how TOSCA enhances evolution management of software as well as address compliance modeling of non-functional requirements. The paper proposes an approach that maximizes involvement of stakeholders in setting up and monitoring TOSCA-based blueprints. A key contribution of the paper is that it brings the focus of a new application of TOSCA in compliance modeling, and how to utilize that within the entire ecosystem of software development and provisioning.

The subsequent sections are structured as follows: Sect. 2 presents a background on non-functional requirements and TOSCA as well as related work. Section 3 introduces the proposed approach and explains with an example on how TOSCA handles modeling of non-functional requirements, and how it fits into our approach. Finally, Sect. 4 summarizes and explains validation of the proposed approach.

2 Background and Related Work

2.1 Non-functional Requirements

E-Type software, which automates human or societal activities and involves real world problem solving [6], must change and continuously adapt to real world requirements [1]. While this evolution is regulated by a feedback system, it typically results in an increase in complexity and decline in quality driven by the need to maintain familiarity [7]. The challenge comes with the objective of controlling the continuous evolution in a systematic way. One solution is to adopt model-driven engineering (MDE), since it allows abstraction of unnecessary details, and to focus on more important aspects (e.g., domain-specific needs) [8]. Another way is to use modeling languages to standardize

software design and improve the management of software evolution [4]. In all this, it is critical to differentiate between functional and non-functional requirements, as they require different tools and skills for modeling let alone the resources needed.

While there is no formal definition or a complete list of non-functional requirements [9], Glinz [9] surveyed existing literature on the definition, classification and representation of non-functional requirements. Their study presents a taxonomy to define non-functional requirements of three categories: performance requirements, specific quality requirements, and constraints. Performance requirements include timing, speed, volume and throughput. Specific quality requirements include reliability, usability, security, availability, portability, and maintainability. Constraints include physical, legal, cultural, environmental, design, implementation, and interface. The international organization for standardization (ISO) [10] however categorizes software quality requirements into eight categories. It does not classify them into functional and non-functional due to overlaps in some requirements. These requirements are functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability. ISO also defined sub-characteristics for each of these requirements. As we focus mainly on non-functional requirements, functional suitability and usability, which are more related to functional requirements of a system are excluded from our discussion.

Among non-functional aspects, which can be modeled using TOSCA, are: (i) enhancement of reliability through scalability thresholds that ensure availability and allow re-instantiating failed components [10, 11]; (ii) improvement of performance and resource utilization [11]; (iii) support of security-by-design (e.g., enforcement of certain encryption mechanisms and access policies) [12]; (iv) increase of compatibility and standardized blueprints [13]; (v) enhancement of maintainability through modularity, reusability, and analyzability of an application [2, 14]; and (vi) ensuring portability and provider-agnostic deployment [12, 15].

2.2 Related Work on Modeling Non-functional Requirements with TOSCA

Many studies in the literature discuss applications of TOSCA in modeling of policies and non-functional requirements. Waizenegger et al. [16] introduced two approaches to model and enforce policies, and provide different levels of abstraction depending on the level of details needed. Built on TOSCA policies and management plans, these approaches focus on providing global knowledge of services as well as enforcement at a component level. They also highlight the importance of reusability of artifacts to minimize the efforts of modeling and provide a wider range of options to customers.

Koetter et al. [17] introduced a Generic Compliance Descriptor, to address the gap between IT and law, linking IT and law to implementation rules that facilitate responses to changes. To do so, they used different technologies at different application life cycles. For example, they collect compliance rules during the design time and link them to the compliance requirements for enforcement during run-time. They used the TOSCA Policy template for modeling security aspects, to ensure that their database and its underlying system is located within the same country. Similarly, and in the context of third-party deployment models, Zimmermann et al. [18] proposed an approach that uses TOSCA, to enforce third-party deployment models to be executed within a company's network. As enforcement of this kind of security policy is critical, third-party applications have to

be enforced to be executed within a company's network, ensuring that vital information does not leave the company [18]. In a slightly wider perspective, Krieger et al. [19] use TOSCA, to automate compliance checking of deployment models with the aim of addressing the issues of changing rules and regulations at the corporate level. Their approach allows separating modeling of compliance rules from modelling of deployment models, so that modelers do not need to know all constraints and requirements to specify compliant deployment models.

Motivated by the growing trend of home-based healthcare, which poses challenges in data collection, transferring, and sharing due to geographical distance between the patients and their care providers, Li et al. [20] apply TOSCA for heterogeneous home-edge-core clouds. They intend to bridge the gap between the availability of software defined infrastructure and meeting regulatory compliance. In the same context, Carrasco et al. [21] introduced a provider-agnostic TOSCA-based model, to allow specification of characteristics and requirements of any system for deployment in the cloud. Besides facilitating the reusability of cloud services, such standardized description of applications, cloud resources, and service APIs can significantly reduce the issues of portability, interoperability, and vendor lock-in.

Despite these works on modeling non-functional aspects of software, exploitation of TOSCA is still under-represented [4]. In addition to that, the extent to which TOSCA can enhance the evolution management of software, is not fully explored.

3 Proposed Approach

3.1 Background on the Workings of TOSCA

While the main purpose of TOSCA is to enhance automation of deployment and management of cloud applications, its functionality can be extended to include modeling and specification of policies, architectural specification of a software service, topology design, service template design, and other non-functional requirements [14]. A TOSCA topology template defines the structure of an application and the orchestration artifacts. While the structure defines application components and the relationships between them, the orchestration artifacts define the deployment and management plans of the application components [16]. Figure 1 shows a topology template for a web application based on OpenTOSCA[1]. The topology describes the components of the application and relationships between them. DjWebApp connects to the DjDB database and depends on Python APIs. The template states that DjWebApp should be hosted on a NGINX server running in a Docker container. DjDB is of type MySQL 5.5 and should be hosted on a separate container. All containers run on a Docker engine hosted on a Linux server of type Ubuntu 18.04. The numbers on each node specify a minimum and maximum number of instances to be created. For example, the AppContainer node can scale up to 10 instances, when the load on the application reaches its peak, and can scale down to 1 instance, if resources are no longer needed. While this description of the topology is at a high level and abstract, detailed specifications of each node and relationship are further elaborated and modeled at a lower level.

[1] https://www.opentosca.org.

A detailed specification of each node and relationship is elaborated using a TOSCA document definition. Such specifications include policies and constraints to be enforced at node and relationship levels. While the TOSCA definition document contains type definitions of Node Types, Relationship Types, Artifact Types, and Policy Types, a TOSCA topology template contains instances of these definitions with assigned values, ready for execution by a TOSCA-compliant orchestrator.

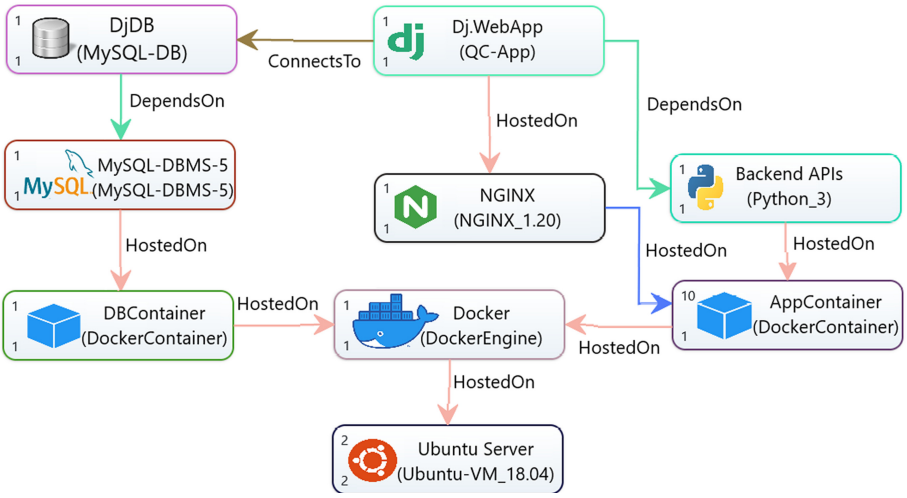

Fig. 1. Topology template example of a web application using OpenTOSCA modelling.

According to OASIS [3], TOSCA can be extended to new types, relationships, policies, and management plans. This allows extensibility of orchestrators' functionality to process these new definitions. The snippet in Fig. 2 shows the syntax of nodes and policy templates in a YAML format. Policy templates define policies and actions to trigger in case of any violation, which in turn enhances the overall reliability and performance. In general, the TOSCA template can serve as a reference architecture with different levels of abstraction. In addition to that, the decomposition of an application into small units along with clear relationships allows for an enhanced evolution management of a software.

3.2 Proposed Architecture for Handling Non-functional Requirements

Software related policies and constraints are mostly the concern of more than one stakeholder [22], who are in charge of different aspects of compliance. The different levels of abstraction that TOSCA provides [14] makes it possible to engage stakeholders of different levels of expertise in the design of software blueprints. The level of abstractions depends on stakeholders' roles and expertise. Preparing a TOSCA blueprint involves stakeholders like IT managers, compliance experts, and software architects. The approach that is presented in Fig. 3 aims at enhancing the evolution management of a software, while controlling compliance to the agreed upon blueprint. The first step is the

```
my_company.my_types.DjWebApp:
   derived_from: tosca.nodes.SoftwareComponent
   properties:
      my_app_password:
         type: string
         constraints:
            - min_length: 6
            - max_length: 10
    attributes:
       web_app_port:
          type: integer
          default: 80, 443
    requirements:
       - Database:
          capability: EndPoint.Database
          node: DjDB
          relationship: ConnectsTo
```

```
my_company_placement_policy:
   type: tosca.policies.Placement.Geolocation
   description: geographic placement of nodes
   properties:
     region: region_endpoint

my_company_scaling_policy:
   type: tosca.policies.scaling
   description: node autoscaling policy
   properties:
     min_instances: 1
     max_instances: 10
     default_instances: 2
   targets: DjWebApp
```

Fig. 2. Example of TOSCA custom definitions of non-functional requirement on the right; and policy definition on the left, based on [3].

development of a TOSCA-based blueprint. This step requires the concerned stakeholders to specify the new policies to model or revise an existing one. The deliverable of this step is a new TOSCA-based blueprint or an updated version. In the second step, the blueprint is stored into the Blueprint Repository, making it available for the development and operations (DevOps) teams to proceed based on that. The DevOps teams are granted only read access on the blueprint so that any fundamental changes at the topology and policy levels have to be reviewed by all stakeholders before deploying them onto production. The third step is to match the active blueprint with the one running in the provisioning. This involves enforcing and auditing the blueprint, and reporting to stakeholders whenever they inquire. Such a task can be performed by extending the functionality of TOSCA-compatible orchestrators (e.g., Kubernetes, which is one of the promising technologies for automating deployment, scaling, and management of applications).

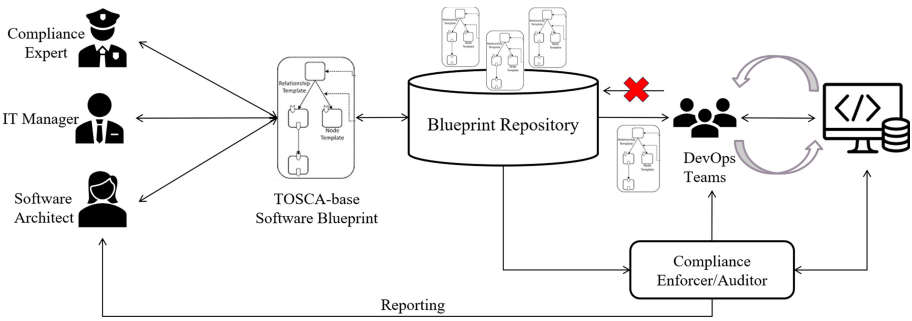

Fig. 3. Proposed architecture for handling compliance of non-functional requirements.

The Blueprint Repository and the Compliance Enforcer/Auditor (Fig. 3) are key components in the proposed approach. The Blueprint Repository stores and keeps track of changes of software blueprints over time through versioning the releases. This enhances reusability of blueprints and simplifies management of the growing complexity of a software. The Compliance Enforcer/Auditor validates and enforces the assigned blueprint

during provisioning. It matches components deployed against the predefined blueprint. If any mismatches are found, the orchestrator stops application provisioning and reports the mismatch right away. As a result, the low-level teams cannot modify the architectural level of the software during the development and provisioning. Changes that require modification on the blueprint topology cannot take place unless a consensus is made among stakeholders on updating the blueprint and, then, pushing it into the repository to be available for enforcement at the production. To keep stakeholders informed, reporting is triggered on the following scenarios: (i) once a new release of the software is made available for production; (ii) upon stakeholders' inquiry on status of the deployed services and how well they align to the blueprint; or (iii) on a regular basis for the purpose of auditing and monitoring depending on corporate policy.

Practically, to keep up with the ever-growing business requirements, the continuing changes and the complexity of an E-type software poses a need for a new way of controlling the evolution and non-functional requirements. While most existing modeling languages focus mainly on functional aspects and the behavior of a software, the proposed approach helps address the non-functional aspects, giving a better visibility of the architectural topology of a software to stakeholders with different levels of abstraction. Distributed software development is a potential application of the proposed approach.

4 Conclusion and Future Work

4.1 Future Validation of the Proposed Approach

The approach proposed needs to be evaluated at technical and process levels. At the technical level, Eclipse Winery[2] or any other TOSCA modeling tool can be used to design a TOSCA-based blueprint and model the non-functional requirements.

Once the blueprint is ready, it has to be validated. TOSCA-Parser[3], which is an OpenStack project, can be extended to parse and validate the blueprint along with newly defined types and policies.

Once validated, a TOSCA-conform runtime environment is needed to deploy and provision the application according to the blueprint. OpenTOSCA Container provides a TOSCA-compliant runtime environment and supports the provisioning of applications. For monitoring and reporting, the TOSCA runtime can be integrated with TOSCA-Parser and extended to allow real time monitoring and reporting of the blueprint being provisioned.

At the process level the approach can be validated through a development of a case with multiple stakeholders collaborating in the setup of a TOSCA-based blueprint. By simulating the steps of the proposed approach, a set of metrics can be developed to evaluate its effectiveness and identify possible improvements.

[2] https://winery.readthedocs.io/en/latest.
[3] https://wiki.openstack.org/wiki/TOSCA-Parser.

4.2 Summary

In this paper, we explored the potential of using the TOSCA standard for modeling non-functional requirements. In particular, we described its potential for compliance specification and enforcement. We also proposed an approach that maximizes involvement of stakeholders in setting up compliance specifications of non-functional requirements in the form of a TOSCA-based blueprint. This blueprint can then be used by DevOps teams as a base and a reference architecture through all stages of the software development life cycle (SDLC). It can also serve as a compliance checking and reporting while provisioning. Moreover, keeping track of changes in topologies over time is expected to give more control over the evolution process of the software. The approach can be useful for managing software projects, which change and grow at a high rate. Examples are cloud native applications, whether on-premise cloud or on clouds.

Besides validating the proposed framework at technical and process levels, as described above, it is planned to extend the application of TOSCA to modeling and specification of other requirements including regulations and industry-specific ones.

Acknowledgements. This research was supported by the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea). This work was also supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2019R1F1A1058487) funded by the Ministry of Science and ICT (MSIT) of Korea.

References

1. Herraiz, I., Rodriguez, D., Robles, G., Gonzalez-Barahona, J.M.: The evolution of the laws of software evolution: a discussion based on a systematic literature review. ACM Comput. Surv. **46**(2), 28:1–28:28 (2013). https://doi.org/10.1145/2543581.2543595
2. Nieuwenhuis, L.J.M., Ehrenhard, M.L., Prause, L.: The shift to Cloud Computing: the impact of disruptive technology on the enterprise software business ecosystem. Technol. Forecast. Soc. Chang. **129**, 308–313 (2018). https://doi.org/10.1016/j.techfore.2017.09.037
3. "TOSCA Version 2.0." OASIS (2020). https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.pdf. Accessed 07 May 2021
4. Bergmayr, A., et al.: A systematic review of cloud modeling languages. ACM Comput. Surv. **51**(1), 22:1–22:38 (2018). https://doi.org/10.1145/3150227
5. Bellendorf, J., Mann, Z.Á.: Specification of cloud topologies and orchestration using TOSCA: a survey. Computing **102**(8), 1793–1815 (2019). https://doi.org/10.1007/s00607-019-00750-3
6. Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proc. IEEE **68**(9), 1060–1076 (1980)
7. Lehman, M.M., Ramil, J.F.: Software evolution and software evolution processes. Ann. Softw. Eng. **14**(1), 275–309 (2002). https://doi.org/10.1023/A:1020557525901
8. Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Softw. Syst. Model. **17**(1), 91–113 (2016). https://doi.org/10.1007/s10270-016-0523-3
9. Glinz, M.: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26, October 2007. https://doi.org/10.1109/RE.2007.45

10. ISO/IEC 25010:2011(en): Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. Accessed 11 June 2021
11. Kim, D., Muhammad, H., Kim, E., Helal, S., Lee, C.: TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform. Appl. Sci **9**(1), Art. no. 1 (2019). https://doi.org/10.3390/app9010191
12. Antonacci, M., et al.: Digital repository as a service: automatic deployment of an Invenio-based repository using TOSCA orchestration and Apache Mesos. EPJ Web Conf. **214**, 07023 (2019). https://doi.org/10.1051/epjconf/201921407023
13. Cankar, M., Luzar, A., Tamburri, D.A.: Auto-scaling using TOSCA infrastructure as code. In: Muccini, H., et al. (eds.) ECSA 2020. CCIS, vol. 1269, pp. 260–268. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59155-7_20
14. Brogi, A., Soldani, J., Wang, P.: TOSCA in a nutshell: promises and perspectives. In: Villari, M., Zimmermann, W., Lau, K.-K. (eds.) ESOCC 2014. LNCS, vol. 8745, pp. 171–186. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44879-3_13
15. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable cloud services using TOSCA. IEEE Internet Comput. **16**(3), 80–85 (2012)
16. Waizenegger, T., et al.: Policy4TOSCA: a policy-aware cloud service provisioning approach to enable secure cloud computing. In: Meersman, R., et al. (eds.) OTM 2013. LNCS, vol. 8185, pp. 360–376. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41030-7_26
17. Koetter, F., Kochanowski, M., Weisbecker, A., Fehling, C., Leymann, F.: Integrating compliance requirements across business and IT. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, pp. 218–225, September 2014. https://doi.org/10.1109/EDOC.2014.37
18. Zimmermann, M., Breitenbucher, U., Krieger, C., Leymann, F.: Deployment enforcement rules for TOSCA-based applications. In: Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018), pp. 114–121 (2018)
19. Krieger, C., Breitenbücher, U., Képes, K., Leymann, F.: An approach to automatically check the compliance of declarative deployment models. In: IBM Research Division, pp. 76–89 (2018)
20. Li, P., Xu, C., Luo, Y., Cao, Y., Mathew, J., Ma, Y.: CareNet: building a secure software-defined infrastructure for home-based healthcare. In: Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, New York, NY, USA, pp. 69–72, March 2017. https://doi.org/10.1145/3040992.3041007
21. Carrasco, J., Cubo, J., Durán, F., Pimentel, E.: Bidimensional cross-cloud management with TOSCA and Brooklyn. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), pp. 951–955, June 2016
22. Rashid, Z., Noor, U., Altmann, J.: Economic model for evaluating the value creation through information sharing within the cybersecurity information sharing ecosystem. Future Gener. Comput. Syst. **124**, 436–466 (2021). https://doi.org/10.1016/j.future.2021.05.033
23. Mohammed, M., Altmann, J.: Software compliance in different industries: a systematic literature review. In: CIISR 2021, International Workshop on Current Compliance Issues in Information Systems Research, March 2021