Modelling and Analysing Cloud Application Management

Antonio Brogi, Andrea Canciani, Jacopo Soldani

Modelling and analysing cloud application management*

Antonio Brogi, Andrea Canciani, and Jacopo Soldani

Department of Computer Science, University of Pisa, Italy

Abstract. Managing complex applications over heterogeneous clouds is one of the emerging problems in the cloud era. The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) aims at solving this problem by providing a language to describe and manage complex cloud applications in a portable and vendor-agnostic way. TOSCA permits to define an application as an orchestration of components, whose types can specify states, requirements, capabilities and management operations — but not how they interact with each other.

In this paper we propose a simple extension of TOSCA that permits to specify the behaviour of management operations and their relations with states, requirements, and capabilities. We show how such an extension permits to automate various useful analyses, like determining the validity of a management plan, which are its effects, or which plans reach certain system configurations. Finally, we illustrate a proof-of-concept graphical interface that permits to edit and analyse management protocols in TOSCA applications.

1 Introduction

Cloud computing has revolutionized IT, by allowing to run on-demand distributed software systems at a fraction of the cost of just a few years ago. However, due to the lack of standardization, how to flexibly manage applications over heterogeneous clouds is still an open issue.

In this scenario, OASIS released TOSCA (*Topology and Orchestration Specification for Cloud Applications* [15,17]), a standard to support the automated management of complex cloud-based applications. TOSCA provides a modelling language to describe, in a portable and vendor-agnostic way, a cloud application and its management. An application is defined by instantiating component types, and by connecting a component's requirements to the capabilities of other components. Its management can then be described by orchestrating the operations of its components (like *configure*, *install*, *start*, etc.) into workflow plans.

Unfortunately, the current version of TOSCA [15] does not permit to specify the behaviour of a cloud application's management operations. More precisely, it is not possible to describe the order in which the management operations of a component must be invoked, nor how those operations depend on the requirements or how they affect the capabilities of that component (and hence

* Work partly supported by the European project SeaClouds (EU-FP7-ICT-610531).

the requirements of other components they are connected to). This implies that the verification of whether a management plan is valid can only be performed manually, with a time-consuming and error-prone process.

In this paper we first propose a simple extension of TOSCA that permits to specify the behaviour of management operations and their relations with states, requirements, and capabilities. We define how to describe the management protocols of TOSCA components by means of finite state machines whose states and transitions are associated with conditions on the component's requirements and capabilities. Intuitively speaking, the objective of those conditions is to define the consistency of component's states and to constrain the executability of component's operations to the satisfaction of their requirements.

We then show how to the proposed extension of TOSCA permits to automate various analyses of management protocols, like determining whether management plans are valid, which are their effects, or which plans permit to reach certain system configurations.

Finally, we illustrate the feasibility of our approach by describing a proof-of-concept web-based application that permits to edit the management protocols of TOSCA application components, and to analyse plans describing the management of a whole application.

The rest of the paper is organized as follows. Sect. 2 introduces TOSCA, while Sect. 3 illustrates a scenario motivating the need for an explicit, machine-readable representation of management protocols. Sect. 4 describes how TOSCA can be extended to model the behaviour of management operations, and how the proposed modelling permits to automate different types of analysis. Sect. 5 illustrates our proof-of-concept. Related work is discussed in Sect. 6, while some concluding remarks are drawn in Sect. 7.

2 Background: TOSCA

TOSCA [15] is an emerging standard aimed at enabling the specification of portable cloud applications and the automation of their management. To do so, TOSCA provides a modelling language to describe the structure of a cloud application as a typed topology graph, and its tasks as plans. More precisely, each cloud application is represented as a `ServiceTemplate` (Fig. 1), consisting of a mandatory `TopologyTemplate` and of optional management `Plans`. Generic type definitions are also contained in the document defining the `ServiceTemplate` as they are referred to by the elements in its topology.

The `TopologyTemplate` is a typed directed graph describing the structure of the composite cloud application. Its nodes (`NodeTemplate`s) model the application components, while its edges (`RelationshipTemplate`s) model the relations among those components. `NodeTemplate`s and `RelationshipTemplate`s are typed by means of `NodeType`s and `RelationshipType`s, respectively. A `NodeType` defines (i) the observable properties of an application component, (ii) the possible states of its instances, (iii) its requirements, (iv) the capabilities it offers to satisfy other components' requirements, and (v) its management op-

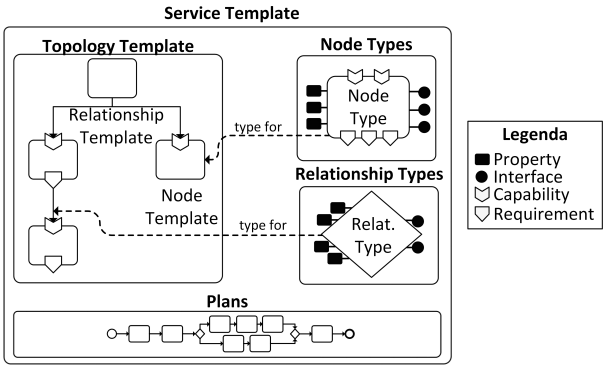

Fig. 1. TOSCA `ServiceTemplate`.

erations. `RelationshipType`s describe the properties of relationships occurring among components. Syntactically, properties are described by `PropertiesDefinition`, states by `InstanceState`s, requirements by `RequirementDefinition`s (of certain `RequirementType`s), capabilities by `CapabilityDefinition`s (of certain `CapabilityType`s), and operations by `Interface`s and `Operation`s.

`Plans` instead allow to describe the management aspects of a `ServiceTemplate`. More precisely, each `Plan` is a workflow orchestrating the management `Operation`s offered by the application components to address (part of) the management of the whole cloud application[1].

3 Motivating scenario

Consider two utility web services, *Translator* and *Convertor*, and suppose that we want to manage them on a TOSCA-compliant cloud platform. After describing the services in TOSCA, we have to specify the third-party application components needed to properly host them. For instance, we may indicate that they have to run on an *Apache* server installed on a *Debian* operating system, which in turn runs on an *VMWare* virtual machine. Fig. 2 illustrates the resulting `TopologyTemplate`, according to the graphical notation introduced by Winery [14]. For the sake of readability, we focus only on the lifecycle interfaces [8] of each `NodeType` instantiated in the topology (i.e., the interfaces containing the operations to install, configure, start, stop, and uninstall a component).

Suppose now that we want to specify the deployment of the *Translator* and *Convertor* services by writing a TOSCA `Plan`. It is worth noting that, since TOSCA does not include any representation of the management protocols of (third-party) `NodeType`s, one may produce invalid `Plans`. For instance, while Fig. 3 illustrates three seemingly valid BPMN `Plans`, only (c) is a valid `Plan`. `Plan` (a) is not valid since *Apache*'s `Configure` operation cannot be executed

[1] A more detailed and self-contained introduction to TOSCA can be found in [8].

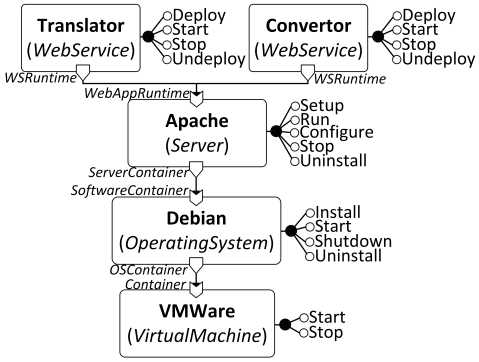

Fig. 2. Motivating scenario.

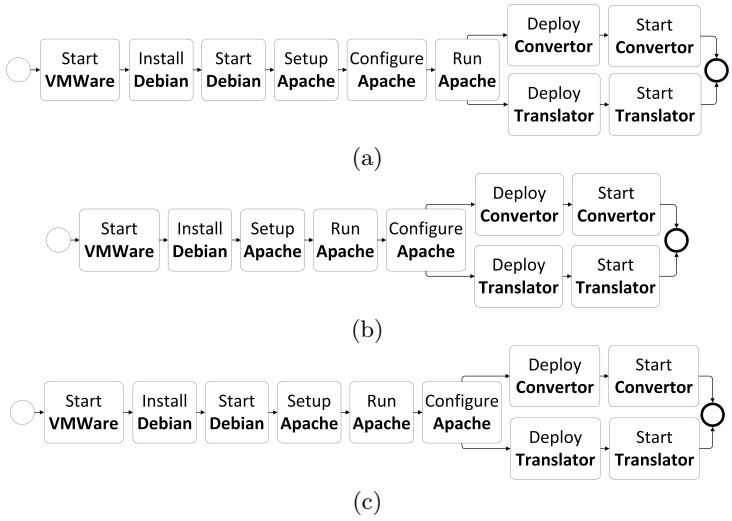

Fig. 3. Examples of deployment `Plan`s.

before *Apache* itself is running, while `Plan` (b) is not valid since *Apache* cannot be installed if the *Debian* operating system is not running.

While the validity of `Plans` can be manually verified, this is a time-consuming and error-prone process. In order to enable the automated verification of the validity of `Plans`, TOSCA needs to be extended with an explicit, machine-readable representation of `NodeType`s' management protocols.

4 Management protocols for cloud applications

TOSCA `NodeType`s can be described by means of their states, requirements, capabilities, and management operations, but there is currently no way to specify

how management operations affect states, how operations or states depend on requirements, or which capabilities are concretely provided in a certain state.

In this section we propose an extension of TOSCA that permits to specify the behaviour of management operations and their relations with states, requirements, and capabilities.

4.1 Definition of management protocols

Let N be a TOSCA `NodeType`, and let us denote its states, requirements, capabilities, and management operations with S_N, R_N, C_N, and O_N, respectively.

We want to describe whether and how the management operations of N depend on (i) other operations of the same node and/or on (ii) operations of other nodes providing the capabilities that satisfy the requirements of N.

(i) The first kind of dependencies can be easily described by specifying the relationship between states and management operations of N. More precisely, to describe the order with which the operations of N can be executed, we introduce a transition relation τ specifying whether an operation o can be executed in a state s, and which state is reached by executing o in s.

(ii) The second kind of dependencies can be described by associating transitions and states with (possibly empty) sets of requirements to indicate that the corresponding capabilities are assumed to be provided. More precisely, the requirements associated with a transition t specify which are the capabilities that must be offered to allow the execution of t. The requirements associated with a state of a `NodeType` N specify which are the capabilities that must (continue to) be offered by other nodes in order for N to (continue to) work properly.

To complete the description, we also associate to each state s of a `NodeType` N the capabilities provided by N in s.

Definition 1. *Let $N = \langle S_N, R_N, C_N, O_N, \mathcal{M}_N \rangle$ be a* `NodeType`*, where S_N, R_N, C_N, and O_N are the finite sets of its states, requirements, capabilities, and management operations. $\mathcal{M}_N = \langle \overline{s}_N, \rho_N, \chi_N, \tau_N \rangle$ is the* management protocol *of N, where*

- *$\overline{s}_N \in S_N$ is the initial state,*
- *ρ_N is a function indicating, for each state $s \in S_N$, which conditions on requirements must hold (i.e., $\rho_N(s) \subseteq R_N$),*
- *χ_N is a function indicating which capabilities of N are concretely offered in a state $s \in S_N$ (i.e., $\chi_N(s) \subseteq C_N$), and*
- *$\tau_N \subseteq S_N \times 2^{R_N} \times O_N \times S_N$ is a set of quadruples modelling the transition relation (i.e., $\langle s, H, o, s' \rangle \in \tau_N$ means that in state s, and if condition H holds, o is executable and leads to state s').*

Syntactically, to represent $\mathcal{M}_N$ we slightly extend the syntax[2] for describing a TOSCA `NodeType`. First, we enrich the description of `InstanceStates` by

[2] A more detailed syntax for extended `NodeTypes` can be found in [5].

introducing the nested elements `ReliesOn` and `Offers`. `ReliesOn` defines ρ_N by enabling the association between states and conditions on requirements, while `Offers` defines χ_N by indicating the capabilities offered in a state. Furthermore, we introduce the element `ManagementProtocol`, to specify the `InitialState` $\overline{s}$ of a protocol, as well as the `Transitions` defining its transition relation τ_N.

The management protocols of the `NodeType`s in our motivating scenario (Sect. 3) are shown in Fig. 4, where $\mathcal{M}_{WS}$ is the management protocol for `WebServices`, $\mathcal{M}_S$ for `Server`, $\mathcal{M}_{OS}$ for `OperatingSystem`, and $\mathcal{M}_{VM}$ for `VirtualMachine`. Consider for instance the management protocol $\mathcal{M}_S$ of the `Server`

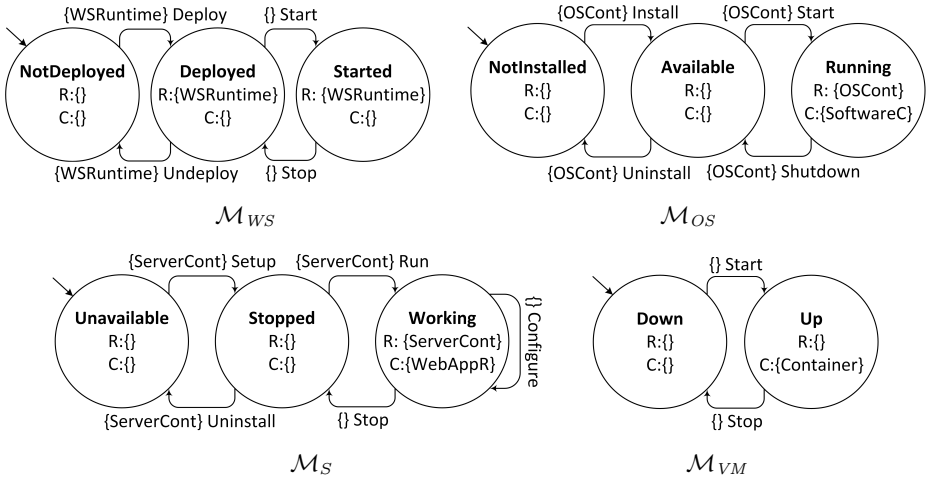

Fig. 4. Management protocols of the `NodeType`s in our motivating scenario.

`NodeType`, typing a *Tomcat* server.Its states S_S are `Unavailable` (initial), `Stopped`, and `Working`, the only requirement in R_S is `ServerContainer`, the only capability in C_S is `WebAppRuntime`, its management operations O_S are `Setup`, `Uninstall`, `Run`, `Stop`, and `Configure`. States `Unavailable` and `Stopped` are not associated with any requirement or capability. State `Working` instead specifies that the capability corresponding to the `ServerContainer` requirement must be provided in order for `Server` to (continue to) work properly. State `Working` also specifies that `Server` provides the `WebAppRuntime` capability when in such state. Finally, all transitions (but those involving operations `Stop` and `Configure`) bind their executability to the availability of the capability that satisfies the `ServerContainer` requirement.

Management protocols (as per Def. 1) allow operations to have non-deterministic effects (e.g., a state may have two outgoing transitions corresponding to the same operation and leading to different states[3]). This form of non-determinism

[3] Note that the conditions of the two transitions may both hold even if the sets of requirements they refer to are disjoint. Hence the state obtained by performing the operation would be non-deterministic.

is not acceptable when managing TOSCA applications [8]. We will thus focus on *deterministic* management protocols (i.e., protocols ensuring deterministic effects when performing an operation in a state).

Definition 2. *Let* $N = \langle S_N, R_N, C_N, O_N, \mathcal{M}_N \rangle$ *be a* `NodeType`*. The management protocol* $\mathcal{M}_N = \langle \overline{s}_N, \rho_N, \chi_N, \tau_N \rangle$ *is* deterministic *if and only if*

$$\forall \langle s_1, H_1, o_1, s_1' \rangle, \langle s_2, H_2, o_2, s_2' \rangle \in \tau_N : (s_1 = s_2 \wedge o_1 = o_2) \Rightarrow s_1' = s_2'$$

4.2 Analysis of management protocols

In this section we describe different analyses that can be performed on the management protocol of a TOSCA application, such as checking the validity of a `Plan`, determining its effects, or discovering `Plan`s that allow to reach certain system configurations.

We first define an *intensional* operational semantics of the management protocol of a single component (viz., a TOSCA `NodeType`), which models all possible sequences of management operations that could be performed on a component if the conditions on the needed requirements were satisfied by the environment. Formally, the intensional semantics of the management protocol of a `NodeType` N can be defined by a labelled transition system over configurations that are the states of N.

Definition 3. *Let* $N = \langle S_N, R_N, C_N, O_N, \mathcal{M}_N \rangle$ *be a* `NodeType`*. The* intensional semantics *of the management protocol* $\mathcal{M}_N$ *of* N *is modelled by a labelled transition system whose set of configurations is* S_N *and where the transition relation is defined by the following inference rule:*

$$\frac{N = \langle S_N, R_N, C_N, O_N, \mathcal{M}_N \rangle \qquad \mathcal{M}_N = \langle \overline{s}_N, \rho_N, \chi_N, \tau_N \rangle \qquad \langle s, H, o, s' \rangle \in \tau_N}{s \xrightarrow{\langle H, o \rangle}_N s'}$$

Intuitively, a transition $s \xrightarrow{\langle H,o \rangle}_N s'$ denotes that operation o can be executed on N when N is in state s, and under the hypothesis that condition H holds, making N evolve into state s'.

The intensional semantics of the management protocol of a single `NodeType` permits to determine the conditions that must hold in the environment for sequences of management operations such as

$$s_0 \xrightarrow{\langle H_1, o_1 \rangle}_N s_1 \xrightarrow{\langle H_2, o_2 \rangle}_N \cdots \xrightarrow{\langle H_h, o_h \rangle}_N s_h$$

to be effectively executable on a `NodeTemplate` of such `NodeType`.

We can now define the semantics of the management protocol of a whole application (viz., a TOSCA `ServiceTemplate`) by suitably composing the intensional semantics of the management protocols of the components (`NodeTemplate`s) that form such application. Formally, the semantics of the management protocol

of a ServiceTemplate S can be defined by a labelled transition system over configurations that denote the states of the NodeTemplates of S. Intuitively, a transition

$$G \xrightarrow{\langle o, N_i \rangle}_S G'$$

denotes that operation o can be executed on NodeTemplate N_i when the "global" state of S is G, making S evolve into the new global state G'.

We first formally define the notion of *global state* of a ServiceTemplate and introduce a shorthand notation to denote the capability connected to a requirement in a ServiceTemplate (e.g., to denote Container as the capability connected to the OSContainer requirement in our motivating scenario — Fig. 2).

Definition 4. *A* global state *of* ServiceTemplate S *is denoted by a set*

$$\{(N_1, s_1), \ldots, (N_m, s_m)\}$$

where $N_1, \ldots, N_m$ *is the set of* NodeTemplates *in* S*, and where* s_i *is a state of* N_i*. We denote by* $\overline{G}$ *the initial global state* S *in which each* NodeTemplate *is in its initial state (viz.,* $\overline{G} = \{(N_1, \overline{s}_1), \ldots, (N_m, \overline{s}_m)\}$*).*

We also denote by $cap_S(r)$ *the (partial) function associating a requirement* r *with the capability connected to* r *in* S *by means of a* RelationshipTemplate*.*

We can now formally define the semantics of the management protocols in a ServiceTemplate S. Intuitively, a management operation o can be executed on a NodeTemplate N_i only if all the requirements needed by N_i to perform o are satisfied by the capabilities provided by (other) NodeTemplates in S.

Definition 5. *The* semantics of the management protocols *in a* ServiceTemplate S *is modelled by a labelled transition system whose configurations are the global states of* S*, and where the transition relation is defined by the following inference rule:*

$$\frac{\begin{array}{c} G = \{(N_1, s_1), \ldots, (N_i, s_i), \ldots, (N_m, s_m)\} \\ G' = \{(N_1, s_1), \ldots, (N_i, s_i'), \ldots, (N_m, s_m)\} \\ s_i \xrightarrow{\langle H, o \rangle}_{N_i} s_i' \qquad \forall r \in H : cap_S(r) \text{ is defined} \wedge cap_S(r) \in \bigcup_{j=1}^{m} \chi_{N_j}(s_j) \end{array}}{G \xrightarrow{\langle o, N_i \rangle}_S G'}$$

Definition 5 permits to model the evolution of a ServiceTemplate when a sequence of management operations is executed:

$$G_0 \xrightarrow{\langle o_1, N_{i_1} \rangle}_S G_1 \xrightarrow{\langle o_2, N_{i_2} \rangle}_S \cdots \xrightarrow{\langle o_h, N_{i_h} \rangle}_S G_h.$$

It is worth observing that while Definition 5 checks that the requirements needed by a NodeTemplate N_i to perform an operation o are satisfied by the capabilities provided by the (other) NodeTemplates in S, it does not check whether *after* performing o the requirements assumed by (the states of) all NodeTemplates will continue to be satisfied. We hence introduce the notion of *consistent* global state of a ServiceTemplate.

Definition 6. *A global state* $\{(N_1, s_1), \ldots, (N_m, s_m)\}$ *of a* `ServiceTemplate` S *is* consistent *if and only if*

$$\forall i \in \{1..m\}, \forall r \in \rho_{N_i}(s_i) : cap_S(r) \text{ is defined} \wedge cap_S(r) \in \bigcup_{j=1}^{m} \chi_{N_j}(s_j).$$

Definitions 5 and 6 allow us to formally characterize the *validity* of a sequence of management operations.

Definition 7. *A sequence* $o_1 o_2 \ldots o_n$ *of management operations is* valid *from a global state* G_0 *of a* `ServiceTemplate` S *if and only if:*

$$G_0 \xrightarrow{\langle o_1, N_{i_1} \rangle}_S G_1 \xrightarrow{\langle o_2, N_{i_2} \rangle}_S \cdots \xrightarrow{\langle o_n, N_{i_n} \rangle}_S G_n$$

and each G_i *is a consistent global state.*

The validity of a TOSCA `Plan` descends immediately from Def. 7.

Definition 8. *Let* G *be a global state of a* `ServiceTemplate` S*. A* `Plan` P *for* S *is* valid *from* G *if and only if all its sequential traces are valid in* G*.*

It is easy to see now that the deployment plan (c) of Fig. 3 is valid since, by starting from the initial global state, all its sequential traces are valid (and reach the same global state). Conversely, `Plan`s (a) and (b) in Fig. 3 are not valid as their traces are not valid. More precisely, `Plan` (a) is not valid since all its sequential traces produce the derivation shown in Fig. 5, and *Apache*:`Configure`

VMWare	Debian	Apache	Translator	Convertor
Down	NotInstalled	Unavailable	NotDeployed	NotDeployed
⇓ **VMare** : Start				
VMWare	**Debian**	**Apache**	**Translator**	**Convertor**
Up	NotInstalled	Unavailable	NotDeployed	NotDeployed
⇓ **Debian** : Install				
VMWare	**Debian**	**Apache**	**Translator**	**Convertor**
Up	*Available*	Unavailable	NotDeployed	NotDeployed
⇓ **Debian** : Start				
VMWare	**Debian**	**Apache**	**Translator**	**Convertor**
Up	*Running*	Unavailable	NotDeployed	NotDeployed
⇓ **Apache** : Setup				
VMWare	**Debian**	**Apache**	**Translator**	**Convertor**
Up	Running	*Stopped*	NotDeployed	NotDeployed

Fig. 5. Initial evolution according to `Plan` (a) in Fig. 3.

cannot be executed in the reached global state (because it requires *Apache* to be in state `Working`, instead of `Stopped`). On the other hand, `Plan` (b) is not valid since all its traces start as shown in Fig. 6, and *Apache*:`Setup` cannot be executed in the reached global state. It indeed requires the capability satisfying

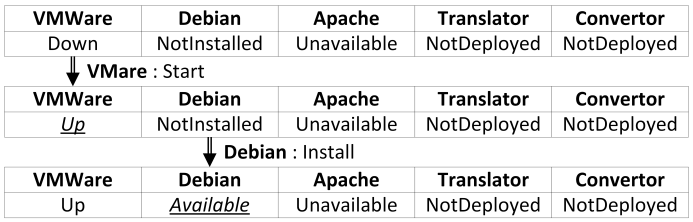

Fig. 6. Initial evolution according to `Plan` (b) in Fig. 3.

Apache's `ServerContainer` to be provided, but that capability is not provided when *Debian* is not in state `Running`.

The introduced modelling can be exploited for various other purposes besides checking `Plan`s validity. For instance, valid `Plan`s may not be enough, as their sequential traces may reach different global states. It is thus interesting to characterize deterministic `Plan`s.

Definition 9. *Let G be a global state of a* `ServiceTemplate` *S. A valid* `Plan` *P for S is* deterministic *from G if and only if all its sequential traces reach the same global state G'.*

It is also interesting to compute the effects of a valid `Plan` P on the states of the components of a TOSCA `ServiceTemplate`, as well as on the requirements that are satisfied and the capabilities that are available. Such effects can be directly determined from the global state(s) reached by performing the sequential traces of P. Moreover, the problem of finding whether there is a deployment `Plan` which starts from the initial global state $\overline{G}$ and achieves a specific goal (e.g., bringing some components of an application to specific states or making some capabilities available) can be solved with a breadth-first search of the reachable global states. The same approach also works in the case of generic management plans (i.e., plans starting from a generic global state G), and it permits to find the sequential `Plan`s (if any) allowing to reach a certain goal from whatever starting G. It also allows to characterize an interesting property that a `ServiceTemplate` may exhibit: if it is possible to reach the intial global state $\overline{G}$ from any G that is reachable from $\overline{G}$ itself, then it is always possible to generate a plan for any (reachable) goal from any (reachable) global state. This ensures reversibility of actions, meaning that whatever G we reach from $\overline{G}$, we can always get back to $\overline{G}$, thus always permitting a (soft) reset of the application.

5 Proof-of-concept implementation

We now illustrate the feasibility of our approach by introducing Barrel, a web-based application[4] that permits to edit and analyse management protocols in

[4] Barrel's interface is written in HTML5, while its back-end is written in JavaScript. The application can be accessed at `http://ranma42.github.io/MProt/` with any modern web-browser, like Google Chrome or Mozilla Firefox. The source code is publicly available at `https://github.com/ranma42/MProt`.

TOSCA applications. In the following, we shall not deepen into implementation details, but rather focus on how Barrel can be used to edit and analyse existing TOSCA applications.

The very first step is to import a CSAR package[5] containing a `ServiceTemplate`, as well as the `NodeType`s instantiated in its `TopologyTemplate`. Once the CSAR is loaded, the `NodeType`s' names appear in the left hand pane of Barrel's interface (Available NodeTypes), and by selecting one of them the user can start editing its management protocol (Fig. 7). The management protocol

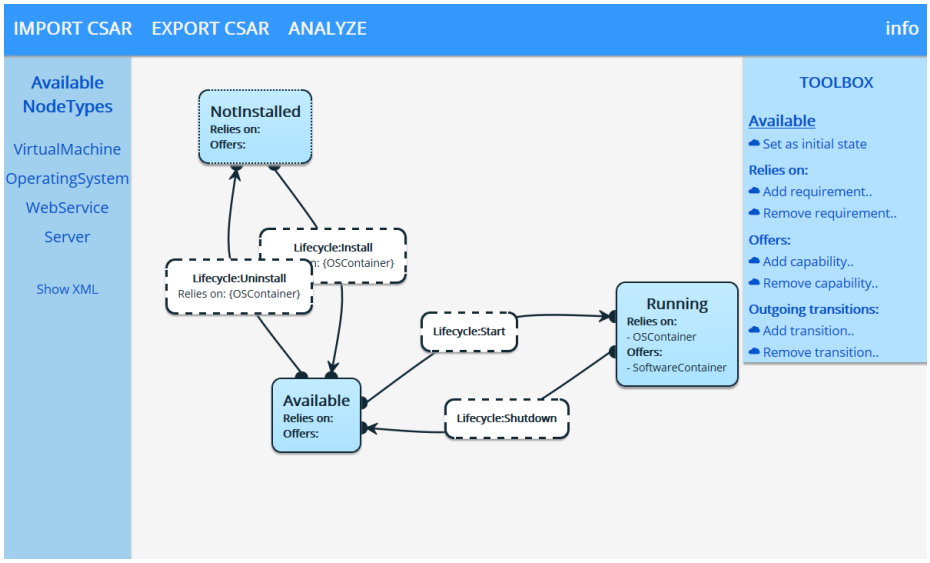

Fig. 7. Screenshot of Barrel: Editing mode.

is visualized in the central pane, by displaying the selected `NodeType`' states and the transitions among these states (if any). By clicking on a state s, a dedicated TOOLBOX opens in the right pane. This TOOLBOX permits editing the current values of $\rho(s)$, $\chi(s)$, and $\tau(s)$, by allowing the user to update the set of requirements on which the selected state s relies, the set of capabilities it offers, and its outgoing transitions. Such updates can also be viewed directly in the XML source of the current `NodeType`, by clicking on the Show XML button in the left pane. Once the `NodeType`s' management protocols have been edited, the updated CSAR can be downloaded through the EXPORT CSAR functionality.

Users can also analyse the behaviour of the management operations appearing in the imported `ServiceTemplate` by selecting the ANALYZE option in the top menu. As a result, Barrel pops out a window showing the current global state of the application topology (Fig. 8). More precisely, the window lists all the

[5] A CSAR (*Cloud Service ARchive*) is a compressed zip file containing the TOSCA definitions describing the cloud application, along with the concrete artefacts implementing its components [15].

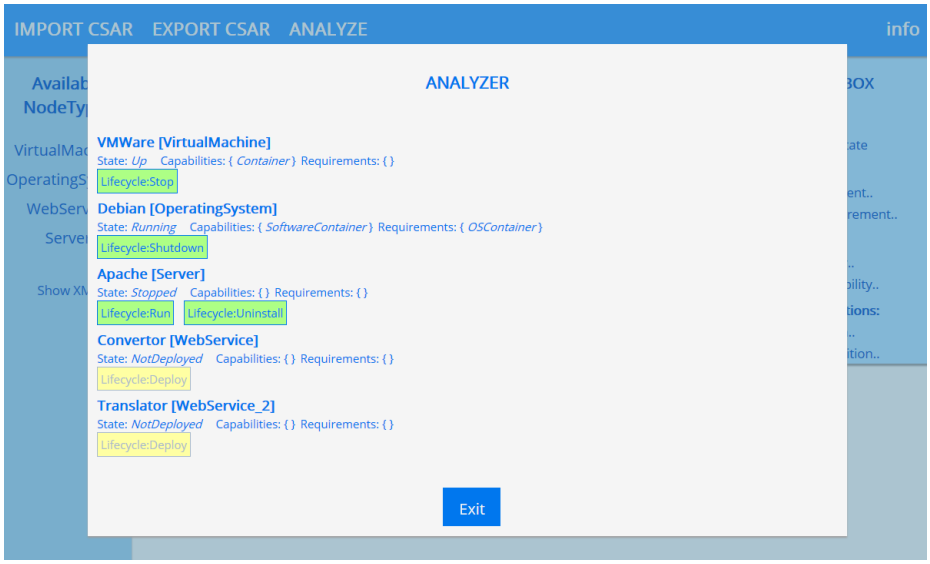

Fig. 8. Screenshot of Barrel: Analysis mode.

`NodeTemplate`s in the `TopologyTemplate`, each associated with its current state, the requirements it relies on, the capabilities it offers and the operation actually available. Each operation is highlighted in green if all the capabilities connected to the requirements needed to execute it are currently available, otherwise it is highlighted in yellow. By clicking on a (green) operation users can simulate its execution, thus updating the current global state and then the ANALYZER window. If the reached state is inconsistent, a warning banner is displayed.

With the simple, interactive ANALYZER of Barrel, users can perform the analyses described in Sect. 4.2. For instance, to check whether a `Plan` is valid, they just need to simulate its sequential traces and check that no inconsistent state is traversed. They can also compute the effects of a valid `Plan` on states, capabilities and requirements by looking at the initial and final configurations displayed by the ANALYZER window. In this first version of Barrel, developers can only perform these analyses interactively, by manually clicking on the (green) operations and by looking at how they affect the global state[6].

It is worth noting that Barrel is already partially integrated with the OpenTOSCA open source ecosystem [3,14]. Barrel is indeed able to process CSARs developed with the visual editor Winery [14], and it produces CSARs that can be imported in Winery[7].

[6] As part of our future work, we intend to extend Barrel in a working prototype capable of automatically performing all the aforementioned analyses.

[7] While Winery imports the CSARs generated by Barrel, it does not properly process the information concerning management protocols. This is obviously because the extension to TOSCA we propose is not yet part of the TOSCA standard, and hence not (yet) supported in the OpenTOSCA open source environment.

6 Related work

The problem of automating application management is well-known in computer science. In the cloud era, it has become even more prominent because of the complexity of both applications and platforms [9]. This is witnessed by the proliferation of so-called "configuration management systems", like Chef [10] or Puppet [18]. These management systems provide domain-specific languages to model the desired configuration for a software solution, and employ a client-server model to ensure that such configuration is met. However, the lack of a machine-readable representation of how to effectively manage cloud application components inhibits the possibility of performing automated analyses on components' configurations and dependencies.

A first attempt to model the deployment of cloud-based applications was the Aeolus component model [11]. The Aeolus model shares our objective of describing various characteristics of cloud applications' components, including the possibility that component interfaces may vary depending on the internal component state. However, the Aeolus model only permits specifying what is offered and required in a state. Our approach instead allows developers to distinguish the requirements ensuring the consistency of a state from those constraining the applicability of a management operation. This permits to express transitions whose requirements concerns only the applicability of an operation and not the consistency of a state (e.g., the transition ⟨`Unavailable`, {`ServerContainer`}, `Setup`, `Stopped`⟩ of the protocol $\mathcal{M}_S$ in Fig. 4). Such kind of transitions cannot be directly modelled in Aelous (without introducing dummy intermediate states). Furthermore, Aelous and other emerging solutions, like Juju [13] or Engage [12], differ from our approach since so far they focus on the *deployment* of a cloud application, rather than on its whole *management*. Aelous, Juju, and Engage also differ from our approach since they are currently not integrated with any cloud interoperability standard.

TOSCA's rich type system has been exploited to devise various techniques that facilitate the the reuse of available services, like [4,7,19]. Those techniques permit to match and adapt (fragments of) existing `ServiceTemplate`s to implement a desired `NodeType` by checking that the features of the latter are all provided by the former. While those techniques are capable of overcoming various syntactical differences, they do not take into account the behaviour of management operations. Namely, they do not check whether the behaviour of a (fragment of) `ServiceTemplate` is compatible with the desired behaviour of a `NodeType`. As our proposal extends TOSCA's type system, it can be naturally exploited to extend the reuse techniques based on TOSCA, like [4,7,19], to account for management behaviour.

Finally, we have investigated the possibility of employing composition-oriented automata (like *interface automata* [1]) to model valid plans directly as the language accepted by the automaton obtained by composing the automata modelling the management protocols of the components of an application. The main drawbacks of such an approach are the size of the obtained automaton (which grows exponentially with the number of application components and

hence makes the automaton scarcely readable even for simple applications), and the need of recomputing the automaton whenever a new component is added or its management protocol is modified.

7 Conclusions

In this paper we have proposed an extension of TOSCA to model the behaviour of management operations and their relations with states, requirements, and capabilities. We have then illustrated how such modelling permits to automate different analyses, such as determining whether a management `Plan` is valid, which are its effects, or which `Plan`s allow to reach certain system configurations. To illustrate the feasibility of the proposed approach, we have developed a proof-of-concept graphical interface that permits to edit `NodeType`s' management protocols and to analyze `ServiceTemplate`s' `Plan`s.

It is worth noting that, even if some of the behaviour-aware analyses discussed in Sect. 4.2 have exponential time complexity in the worst case, they still constitute a significant improvement with respect to the state-of-the-art, as currently the development and validation of `Plan`s is performed manually, after delving through the documentation of the application's components.

It is also worth observing that our approach builds on top of, but is not limited to, TOSCA. It can indeed be adapted to other languages for specifying cloud applications (e.g., like CAMP [16] or GENTL [2]), and more in general to any stateful behaviour model of systems that describe states, requirements, capabilities, and operations.

We are currently investigating the possibility of modelling management protocols for cloud-based applications with Petri nets [6], with the objective of expressing some of the analyses described in Sect. 4.2 in terms of well-known Petri net notions (e.g., expressing `Plan`'s validity in terms of firing sequences, or reducing `Plan` determination to coverability) and hence to possibly exploit some of the many available tools supporting the analyses of Petri nets. We see two other directions for immediate future work. On the one hand, we intend to extend our proof-of-concept Barrel into a working prototype supporting all the analyses described in Sect. 4.2, and to fully integrate it with the OpenTOSCA open source environment [3,14]. On the other hand, as we anticipated in Sect. 6, another interesting direction for future work is to extend the matching and adaptation reuse techniques based on TOSCA [4,7,19] to take into account the management behaviour of cloud-based applications.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 109–120. ESEC/FSE-9, ACM (2001)

2. Andrikopoulos, V., Reuter, A., Sáez, S.G., Leymann, F.: A GENTL Approach for Cloud Application Topologies. In: Villari, M., Zimmermann, W., Lau, K.K. (eds.) Service-Oriented and Cloud Computing. LNCS, vol. 8745, pp. 148–159. Springer (2014)
3. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing. LNCS, vol. 8274, pp. 692–695. Springer (2013)
4. Brogi, A., Soldani, J.: Matching cloud services with TOSCA. In: Canal, C., Villari, M. (eds.) Advances in Service-Oriented and Cloud Computing, CCIS, vol. 393, pp. 218–232. Springer (2013)
5. Brogi, A., Canciani, A., Soldani, J.: Modelling the behaviour of management operations in TOSCA. Tech. Rep., University of Pisa (July 2015)
6. Brogi, A., Canciani, A., Soldani, J., Wang, P.: Modelling the behaviour of management operations in cloud-based applications. In: Moldt, D. (ed.) Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'15. CEUR Workshop Proceedings, vol. 1372, pp. 191–205. CEUR-WS.org (2015)
7. Brogi, A., Soldani, J.: Reusing cloud-based services with TOSCA. In: INFORMATIK 2014, Lecture Notes in Informatics (LNI). vol. 232, pp. 235–246. Gesellschaft für Informatik (GI) (2014)
8. Brogi, A., Soldani, J., Wang, P.: TOSCA in a Nutshell: Promises and Perspectives. In: Villari, M., Zimmermann, W., Lau, K.K. (eds.) Service-Oriented and Cloud Computing. LNCS, vol. 8745, pp. 171–186. Springer (2014)
9. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599 – 616 (2009)
10. Chef: Opscode. `https://www.opscode.com/chef`
11. Di Cosmo, R., Mauro, J., Zacchiroli, S., Zavattaro, G.: Aeolus: A component model for the cloud. Information and Computation 239(0), 100 – 121 (2014)
12. Fischer, J., Majumdar, R., Esmaeilsabzali, S.: Engage: A deployment management system. In: Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 263–274. PLDI '12, ACM (2012)
13. Juju: DevOps distilled. `https://juju.ubuntu.com`
14. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing. Springer (2013)
15. OASIS: Topology and Orchestration Specification for Cloud Applications. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf` (2013)
16. OASIS: Cloud Application Management for Platforms (CAMP). `http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf` (2014)
17. OASIS: TOSCA Simple Profile in YAML. `http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf` (2014)
18. Puppet: Puppet labs. `https://puppetlabs.com`
19. Soldani, J., Binz, T., Breitenbücher, U., Leymann, F., Brogi, A.: TOSCA-MART: A method for adapting and reusing cloud applications. Tech. Rep., University of Pisa (March 2015)