SICS Software-Intensive Cyber-Physical Systems (2019) 34:163–172
https://doi.org/10.1007/s00450-019-00404-x

SPECIAL ISSUE PAPER

TOSCA-based *Intent* modelling: goal-modelling for infrastructure-as-code

Damian A. Tamburri[1] · Willem-Jan Van den Heuvel[1] · Chris Lauwers[2] · Paul Lipton[3] · Derek Palma[4] · Matt Rutkowski[5]

Published online: 13 February 2019

Abstract
DevOps entails a set of practices that speed up the time needed to rollout software product changes. One such practice is automating deployment and delivery with infrastructure-as-code, i.e., automated scripts that ideally carry out 1-click deployment. Providing effective infrastructure-as-code poses the tricky issue in determining the modelling and information representation paradigm (e.g., Imperative, Declarative, etc.) most compatible with specifying infrastructural code. The OASIS TOSCA standard ("Topology and Orchestration Specification for Cloud Applications") is the de-facto and de-iure standard language for infrastructure-as-code, and adopts an innovative take called "intent modelling". This paper articulates the foundations of this modelling approach incorporating the most related modelling paradigm, that is, goal-modelling. We elaborate on it with a real but simple industrial sample featuring the TOSCA language.

Keywords DevOps · Infrastructure-as-code · Orchestration · Microservices · TOSCA · Goal-modelling

1 Introduction

According to Len Bass' seminal book [1], DevOps means meshing software development and operations processes together by augmenting them with any set of practices that accelerate software product changes rollout—infrastructure-as-code being one such practice. This paper elaborates on *intent modelling*, a way to design infrastructure-as-code which is both more generalisable and effective for orchestration, that is, the process of integrating two or more applications and/or services together in an automated fashion, or synchronize data and application management in real-time [2]. We illustrate intent modelling elaborating on its foundational aspects and characteristics, as distilled while designing and experimenting with TOSCA, the *"Topology and Orchestration Specification for Cloud Applications"*, the de-facto and de-iure industrial standard for infrastructure-as-code (IasC).

In our standardization endeavour, we noticed that, on the one hand, IasC promises to deliver fast and painless continuous deployment using appropriate continuous deployment scripts and similar automation mechanisms—think of Docker[1] or Chef.[2]

On the other hand, IasC and its effective orchestration require models and descriptions which are capable of providing abstract, structural, and behavioral aspects being generic enough to encompass not only how the infrastruc-

✉ Damian A. Tamburri
d.a.tamburri@uvt.nl ; damianandrew.tamburri@polimi.it

Willem-Jan Van den Heuvel
wjheuvel@uvt.nl

Chris Lauwers
lauwers@ubicity.com

Paul Lipton
Paul.Lipton@ca.com

Derek Palma
dpalma@vnomic.com

Matt Rutkowski
mrutkows@us.ibm.com

1 Jheronimus Academy of Data Science-TU/e, Eindhoven, The Netherlands

2 Ubicity Corp., Santa Clara, USA

3 CA Technologies, New York, USA

4 Vnomic Corp., New York, USA

5 IBM Corp., Mountain View, USA

[1] https://www.docker.com/.

[2] https://www.chef.io/chef/.

ture should behave, but also service property specifications that allow the orchestrator to "substitute" services while maintaining service properties—such specifications closely relate to what is known as goal-modelling [3], a revolutionary paradigm for requirements engineering focusing on logically-related abstractions that gradually reduce the abstraction-level among software requirements of any kind, and with respect to any constraint. In the context of TOSCA, the same concept applies. For example, imagine a running infrastructure relying on external service providers for critical services. An imperative programming of infrastructure-as-code would force infrastructure designers to elaborate all possible state-changes in the projected behavior for that infrastructure while a declarative approach would describe the logic of the structure rather than state-changes or control-flow.

The research question we are exploring in this article is defined as follows: "what kind of infrastructure-as-code (IasC) description (e.g., either declarative or imperative) fits with the specification power achieved by the TOSCA standard?". For example, what available descriptive frameworks can successfully abstract property statements such as: "when Service S1 goes down it needs to be substituted within 10 s no matter what the underlying provisioning is". From the TOSCA industrial perspective [4], we compared the series of statements as above that TOSCA grammar covers with state of the art requirements and specifications engineering frameworks and found that TOSCA is indeed close to Goal-Modelling, namely, the decompositional and incremental subdivision of higher-level goals into more concrete sub-goals [3]. Conversely, TOSCA adds an *intent* connotation, namely, a specific goal whose *sub-goals* need to be satisfied whatever the costs or consequences over the sub-goals. While on one hand, goal-modelling aims at "keeping all goals satisfied" [5], intent-modelling aims at keeping the highest-level goal (i.e., the *intent*) satisfied by consistently changing, tuning, substituting, or adapting any sub-level goal.

The above contribution is beyond the current state of the art which has currently focused on synthesizing the TOSCA View over IasC for cloud applications [6], or exploring specific instances of TOSCA in multiple domains (e.g., software-defined everything [7] or service discovery [8]) and comparing between different TOSCA implementations [9].

We conclude that IasC specifications reflecting the above *intentional* semantics are critical for resilient and long-lived orchestrations. Stemming from these premises, we elaborate on intent-modelling focusing on how the concept came about, its key properties, and limitations, illustrating with an industrial example.

1.1 Structure of the paper.

The rest of this paper is structured as follows. Section 2 elaborates on common modelling paradigms such as declarative [10] or imperative modelling [11], and elaborates on their relation with respect to intent modelling. Section 3 elaborates on intent modelling key characteristics and design principles while Sect. 5 illustrates those characteristics and principles using real-life scenario examples. Finally, Sects. 6 and 7 discuss our results and conclude the paper elaborating on future work.

2 Why intent modelling?

In code-centric software development, attention is commonly laid on one main artefact: code. Functional correctness, extra-functional guarantees, even the quality of the development process is assessed on the produced code. The focus is simply on the "how" the software achieves its goals. Although crucial, focusing on the "how" implies specificity and bias towards specific solutions. In the era of cloud-based systems of systems, IoT (Internet-of-Things) and in general loosely coupled heterogeneous and flexible configurations of systems where XaaS (Everything as a Service) represents the common denominator, the "how" is only a small piece (not as crucial as one would believe) of the information needed for enabling intra- and inter-system communication and collaboration.

2.1 Intent- versus goal-modelling

The previously introduced conjecture around the necessity of intentional or *intent* modelling shares foothold in many software-related disciplines, such as requirements engineering [12]. Goal modelling, for instance, aims at representing, besides "what" the system shall do, even "why" a certain functionality is needed, and "how" it could be implemented [13].

On the one hand, software development is still lagging behind in this respect, probably also due to the strained hunt for ever slimmer and more agile development processes.

On the other hand, several specific software development paradigms, such as model-driven service- and agent-oriented software development in their agile variants, and especially paradigms stressing collaboration among cross-role stakeholders, such as DevOps, have what it takes and can tremendously benefit from embracing the "what" and "why" to complement the "how" in software development.

More specifically, in the context of cloud applications, we observed that an augmented version of goal-modelling, the so-called *intent modelling* [14], is pivotal for enforcing qual-

ity of description in cloud software development in general, in particular in the context of DevOps lifecycle scenarios.

2.2 Intent defined

We refer to intent(ion)s and intentionality in this context as the characteristic which is commonly ascribed to humans in social sciences and social networks research, namely, *Intention* is a mental state (hence, an abstract representation of a situation and its interpretation) that represents a commitment to carrying out an action or actions in the future [15]. Intention involves abstract mental activities such as planning, forethought, and situational awareness [16].

TOSCA gives the above notion of intent an engineering applicability connotation [17].

In more formal terms, a TOSCA topology template defines a single intent I in the scope of the template. Intent I is, in turn, obtained by the coordinated intents of single nodes in the template. Therefore:

$$I(T) = \cup\, I\ (\text{Node}_{1\ldots n}); \tag{1}$$

Where I(T) is the intent hardcoded in the blueprint for service template T, while I(Node_x) is the intent defined in the scope of the node type definition for node Node_x and, finally, X is the range of nodes in blueprint T.

As previously defined in Sect. 2.1, the notion of intent does not aim at "Keeping All Objectives Satisfied" (as originally defined for goal modelling). Rather, the key goal of TOSCA-based intent modelling is that of satisfying the highest-level goal I(T) in Eq. 1. This corollary also means that, while satisfying I(T) is vital, keeping sub-intents for nodes Node_x is *not*.

> **[Definition]** *Intent modelling* Intent modelling entails modelling infrastructure blueprints by specifying a highest-level goal to be satisfied, regardless of how sub-level intents or goals are satisfied.

3 Towards intent modelling: design paradigms compared

3.1 Imperative design

The first evolution step in modelling and implementing software systems essentially consisted in moving from imperative to declarative programming where a number of abstraction devices (e.g., models) hide away implementation details and allow more creative software design reasoning. With the onset of complex cloud applications and the need for their long-lived infrastructure management, this evolution process is augmented with requirements and operational dynamics connected to complex distributed services orchestration, i.e., the automated arrangement, coordination, and management of computer systems, middleware, and services [18]. Also, complex cloud applications forced the adoption of DevOps [1,19] and infrastructure-as-code, i.e., specifying infrastructure blueprints as if they were software code to be versioned, structured according to specific design-patterns, etc. This is where intent modelling comes in. The inception of intent modelling refers to the inability of imperative and declarative approaches to fully express not only the structure and behavior of complex cloud applications but their intended steady-state, i.e., the general behavior and structure of the application, its related middleware and their internal status. To grasp the motivation behind intent modelling, let us walk through the evolutionary steps that led to intent models in the first place.

For example, imagine imperatively addressing the modelling and execution of a highly-distributed modern service-based solution. An imperative approach would essentially make highly-distributed orchestration unfeasible: there would be way too many variables to control or possible states to foresee, not to mention concurrency or similar issues. Figure 1 shows a simple imperative model expressed in BPMN [20]: whatever behavior is expressed in the model is licit, given certain pre- and post-conditions.

3.2 Declarative design

Conversely, from a declarative perspective, specification approaches have at least two possible foci: (a) focus on the structure of what needs to be (re-)deployed; (b) focus on the behavior (or functionality) that needs to be maintained no matter what, assuming that an orchestrator can create an appropriate service structure that guarantees that behavior.

On one hand, in (a) a classical declarative model could be extended to take into account that it may not be possible to "create" entities that reflect the proposed orchestration model, but an orchestrator may require additional services to go through a set of state changes in order to achieve what the model specifies. In terms of service orchestration, this is called "desired state model" or "steady-state model", an extension of "declarative models" that focus exactly on the service state that an infrastructure shall maintain regardless of the how that desired state is actually maintained. For example, the model can contain annotations that allow generalisable substitution and adaptation based on semantic-similarity principles [21].

On the other hand, (b) is a policy-based approach, where desired behavior is expressed in terms of parameters that need to fit within a set rules and constraints. A classical declarative model for this policy-based approach could be

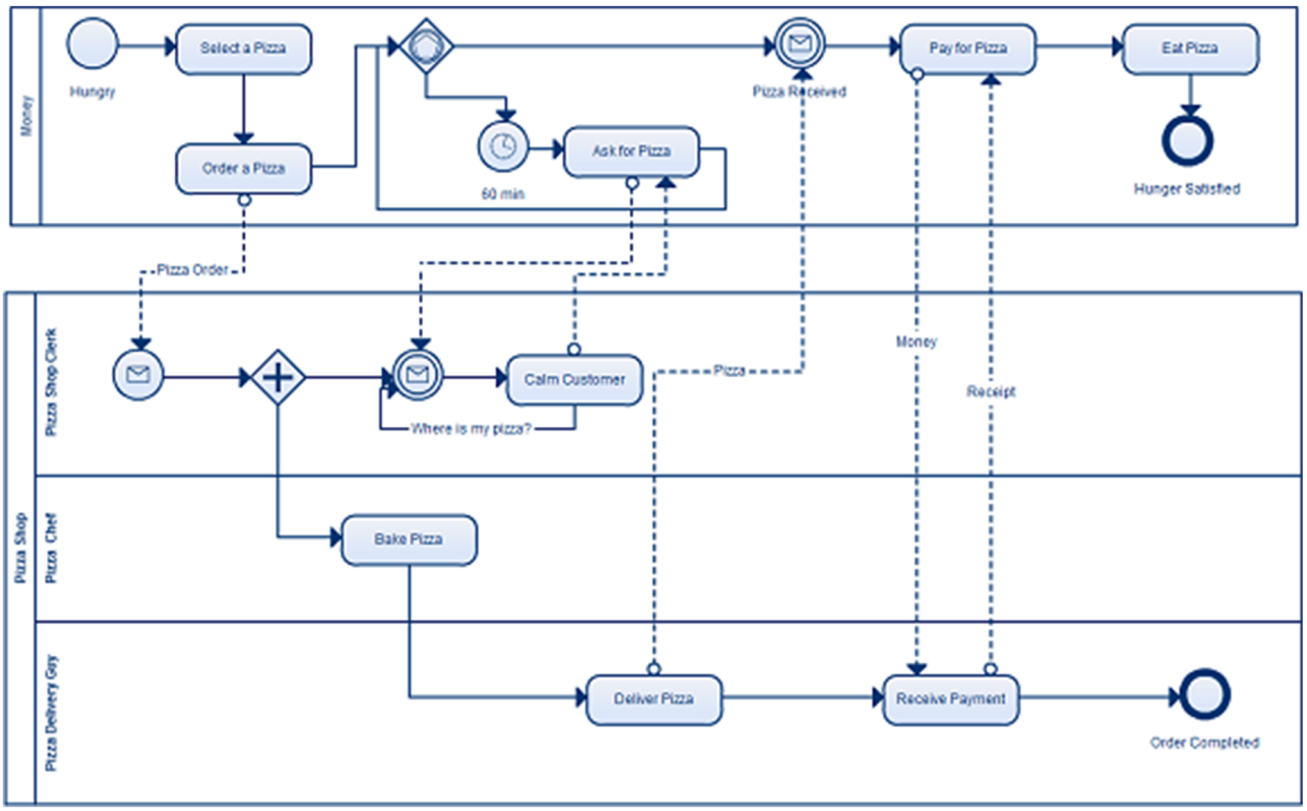

Fig. 1 Imperative modelling example, BPMN; data-flows and services involved are modelled and executed exactly as modelled

extended to a point where requested behavior and functionality can be expressed in a flexible manner using natural language constructs rather than complex policy languages. This approach also presumes some type of mapping service inside the orchestrator that translates "intent"[3] expressions into software-consumable constructs. For example, the Cisco APIC policy infrastructure controllers approach follows this general orchestration approach.

3.3 Intent design

As defined in Sect. 2.2, TOSCA and intent models posit a third, more generic, and hence **standardisable**, option – *intent design*.

Under the assumptions and definitions previously introduced in Sect. 2.2, a single modelling notation can combine the power of expression of (1) policy-based approaches with the modelling power of (2) desired *steady-state* approach.

In turn, the steady-state of an intent design (e.g., a Topology template in the TOSCA parlance) is defined as the length of operational time in which the same desirable intent is maintained, that is, in TOSCA terms, the time that the topology template is maintained operational.

The above combination of (1) and (2) is realised within TOSCA by keeping the template blueprint description *topological*, that is, concerned with the properties of space (topology nodes and edges) that are preserved under continuous deformations (external forces such as scale, backpressure, and more). At the same time, the template blueprint description is kept abstract enough to allow isolation of steady-state model parts, i.e., functional areas of the topology that may be substituted as long as required relations are maintained.

Finally, the notation stemming from this combined approach uses support policies that predicate on the above substitutable elements, as well as the substitution process itself.

3.4 TOSCA-based intent design

As part of our 5+ years' experience in chiselling and refining TOSCA and experimenting with it in our own industrial scenarios, we now begin to understand that the right balance between the above combined elements (1) policy-based and (2) desired steady-state modelling was achieved. The TOSCA Yaml 1.1 specification[4] intermixes both almost equally, essentially by providing basic constructs and means

[3] http://tinyurl.com/j5vkxee.

[4] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/.

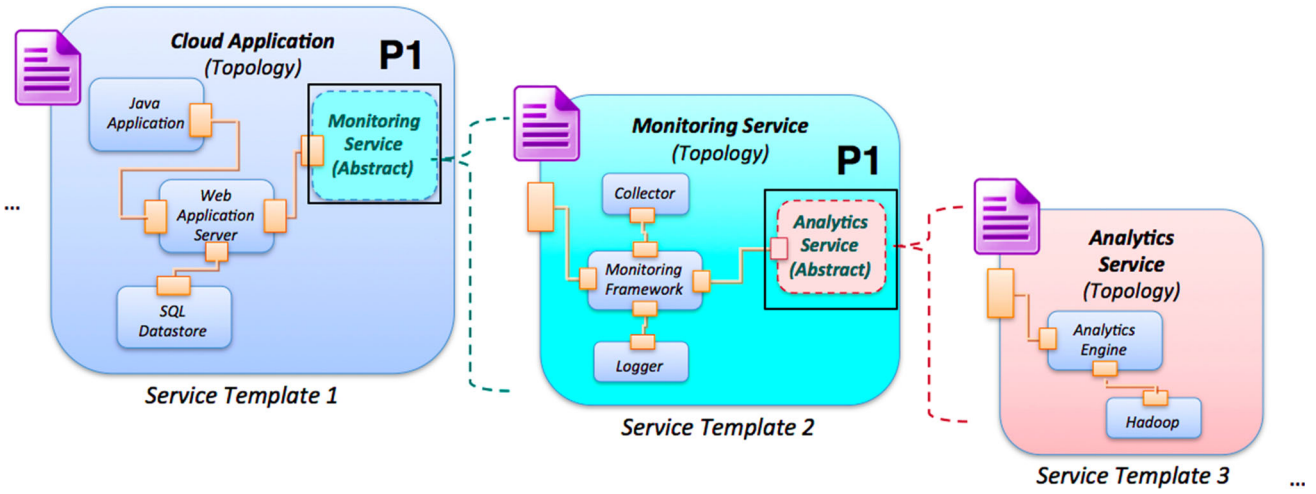

Fig. 2 TOSCA-based substitutability, i.e., topology intent and steady state is maintained as long as capabilities and requirements matching are upheld

to specify entire cloud application infrastructure-as-code using topology templates, i.e., sets of strongly-typed nodes related with strongly-typed relationships [22,23].

In relation to the intent modelling concepts defined previously, the TOSCA standard underlies the following modelling conjecture:

> *The intent hard-coded in a TOSCA specification is maintained as long as types of nodes are consistent, and relationship-to-capability matchings are maintained.*

In other words, as long as there is strong type-consistency and there is a capability that matches each relationship requirement in the TOSCA topology template, then the intent or steady-state encoded in the TOSCA blueprint can be maintained by the orchestrator consuming that blueprint, *indefinitely*.

Figure 2 outlines a simple example of the above intent modelling conjecture and its substitutability property (marked as P1 in the figure, and recurring twice in the illustration). Subsequently, Fig. 3 features an illustrative case featuring the monitoring tier known as ELK (ElasticSearch[5]/Logstash[6]/Kibana[7]). In a TOSCA topology, the ELK can be part of an intent model that includes three tiers, two application tiers and a monitoring tier. Whether or not an ELK tier is available, the intent of the model hard-coded into the blueprint can be maintained as long as the "collectd" and "rsyslog" requirements (lower-left part of Fig. 3, highlighted in red) are met.

TOSCA-based intent modelling features six distinctive characteristics that we distilled by direct experience (reported on Table 1) reflecting: (a) intent models as hierarchies of highly composable and policy-based (micro-)services (see P1 and P6 on Table 1); (b) intent design and implementation as a top-down specification processes (see P3 and P4 on Table 1); (c) microservice symmetry and idempotence as a paramount best practices for good-quality IasC specifications (see P2 and P5 on Table 1).

Assuming that the above modelling properties hold, it follows that any TOSCA-enabled node template (i.e., the instance of a node that can be deployed as part of a topology) can also be substituted with a "compatible" topology, i.e., a TOSCA topology that provides the necessary relationship requirements of the node it is supposed to substitute. Figure 2 provides an example for this feature: a cloud application is essentially a set of node types with corresponding node relationships; nodes can be substituted as many times as technologically possible, as long as capabilities and requirements matching are maintained, that is, as long as *topology intent* is maintained. Subsequently, TOSCA policies are *Event->Condition->Action* statements used to control or factor out undesirable behaviors, e.g., based on specific desired QoS characteristics.

4 TOSCA in pills

TOSCA is designed to express the kinds of entities, relationships, constraints and semantics encountered across the diverse IT domains [4]. TOSCA Service Templates are reusable and composable models that can increase return on investments by making it easy to deploy, configure, and operationally manage more valuable and complex applications throughout their lifecycle from TOSCA models of existing applications and services.

[5] https://www.elastic.co/products/elasticsearch.

[6] https://www.elastic.co/products/logstash.

[7] https://www.elastic.co/products/kibana.

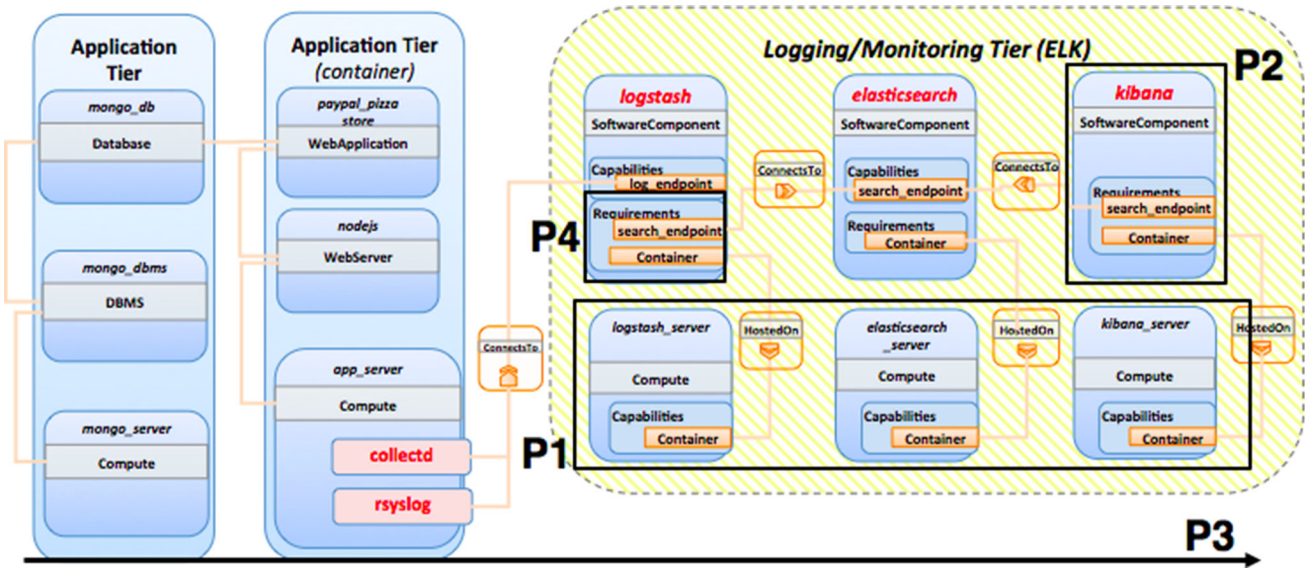

Fig. 3 TOSCA-based intent modelling, i.e., modelling abstract node templates connected with typed relations that refer to relationship types

TOSCA Service Templates describe the overall application/service topology, and consist of a graph of Node Templates that model the application/service components and Relationship Templates that model the relationships and cardinalities between those components, e.g., various kinds of cloud consumer-provider dependencies such as containment and connection. Policy Templates specify the necessary operational constraints on specific parts of the application/service topology.

Furthermore, TOSCA provides a type system to concisely express the schema of the template entities, e.g., node types, relationship types, requirement types, capability types, artifact types and policy types). TOSCA orchestrators and tools process and validate Service Templates by referring to the respective type for each template entity. Naturally, essential types expressing fundamental concepts such as storage, network, and compute are defined and extended to higher-level abstractions, e.g., BlockStorage, Webserver, application and infrastructure components, and much more.

5 Illustrative example

In Figs. 2 and 3 we can see how intent modelling concepts of substructural hierarchy (P1), symmetric idempotence (P2), top-down intentionality (P3) and higher-order scope (P4) are provided in TOSCA.

5.1 Substructural hierarchy

Let us consider P1 in Fig. 2. In the topology defining the sample Cloud Application (Service Template 1), we can find an abstract Monitoring Service in a descriptive form; that is to say at that hierarchical level there is no prescribed way on the "how" that service shall be materialised, but rather a prescribed definition of the capabilities and requirements such a materialisation shall match to maintain the modelled intents. Service Template 2 provides one of the possible valid materialisations of the Monitoring Service. Also in this topology, substructural hierarchy is reveals itself when modelling Analytics Service. Similarly, in Fig. 3 we can see a sub-structure of hierarchies defined at design time to specify a purpose.

5.2 Symmetric idempotence

in the case of idempotence, more specifically the case of interchangeability (Corollary 1), is implied by the reasoning above. In fact, any of the topologies is meant to be replaced by one materialisation at runtime, but many materialisations are viable for each of them. Take the case of ELK in Fig. 3; Kibana is the designated materialisation plugin for data visualisation, but, in principle Grafana[8] could serve the purpose equally well being Kibana and Grafana essentially interchangeable.

5.3 Top-down intentionality

is shown in Fig. 3 where the infrastructure designer is driven into specifying intentions from the top-most generic TOSCA service template (Application Tier) to the most specific ones (ELK) – runtime intent models address realisation which becomes then the responsibility of the orchestrator. In turn, requirements shown in Fig. 3 specify functionality to be addressed by whatever means. Requirements are

[8] http://grafana.org/.

Table 1 TOSCA-based intent modelling concepts, an overview

Name	Description	
P1: Substructural hierarchy	Runtime intent models define a model hierarchy where abstract and functional aspects of IasC blueprints can be decomposed (based on defined policies) to one of a number of possible substructures, all the way down to the point that what you are defining is the deployment of an actual application. Conversely, design time intent models convey the purpose or intent that IasC blueprints entail, regardless of their runtime counterpart	
P2: Symmetric idempotence	Two different intent models are not interchangeable unless they were built from the same template and do not extend that template	
	Corollary 1, interchangeability	An intent model is used based on its exposed properties, anything that exposes exactly the same properties is (normally) interchangeable. This means that if you define a template for the functional class "firewall" and if everything that implements firewall conforms to that intent model template, you could presume them all to be interchangeable and your approach to be open
	Corollary 2, super-structural Substitutability	If you define two intent models for "firewall" that do not expose the same properties, then obviously you cannot substitute one for the other, though you might be able to create a super-model above and harmonize the two different implementations
P3: Top-down intentionality	Application and infrastructure services design starts from features coded into intent models that describe these features, then look down a layer at a time to define new structures that eventually implement them on suitable resources. If you follow this approach, then you are able to map what you are doing to the TOSCA standard	
P4: Higher-order scope	An intent model channels a higher-order purpose, e.g., achieving an ultimate QoS result or elastically performing a service. Intent models allow to specify services as generic computations with a larger scope compared with typical policy-definition services. For example, a TOSCA intent model (i.e., a TOSCA blueprint) can express both result and operation of computation: if a result is described by an intent model, specific actions that show how to achieve that intent may be omitted in the blueprint. Similarly, if an operation is described by an intent model, conditions of operation may be left optional	
P5: Meta-centric design	The key approach behind intent modelling is Going "Meta-". Specifying the "what" metadata is the intent modelling activity itself. The "how" metadata is the corresponding fulfillment and provisioning behavior during service-level automation (i.e., the orchestrator's responsibility). Good quality infrastructure-as-code allows the "what" and "how" to be designed and expressed both in modular packages, which are reusable by assembling higher-level intents based on lower-level components - this approach facilitates the service agility which our industries are looking for	
P6: Resource-centric intent unfolding	The challenge of intent models is that you eventually have to deliver on the intent, which means that you have to be able to decompose the "intent" into realization at the resource level. More in particular, a "service" can set policies within its intent model that would guide the orchestrator into proper resource selection, meaning that a service could require other services preferring one implementation or technology over another, regardless of technical qualities	

themselves consistent with TOSCA Policies which specify a higher-order scope (P4). A final remark concerns the overall assumption by which, in TOSCA, intent means that relationship-to-capability mapping must remain constant: whatever the assumptions behind Fig. 3, the capabilities on the left-hand and middle side in the same example need to be fulfilled, e.g., by instrumenting the substitutability and substructural hierarchy outlined in Fig. 2.

6 Discussions and challenges

In our effort of eliciting the vision conveyed in this paper, we made two key observations.

First, depending on what constructs and abstractions are needed in TOSCA specifications, it would be interesting to look into possible extendibility and refinement among different intent models in their dual nature, i.e., either design or runtime. Users might for instance have several levels of "intent" specification, e.g., partial intents, sub-intents and so on. However, this feature to potentially provide partial specifications is reminiscent of goal modelling [24,25]. Research should be invested in defining intent models more formally such that a more specific mapping of their features, e.g., with respect to goal models can be inferred.

Second, there seems to be an obvious pervasiveness characteristic to be tracked and acted upon for intent models. More specifically, intents have to be mapped with actionable operations at some point in the orchestration workflow. At that point, different options can be taken into account. The two obvious extremes are: (1) intents are the top level entity and are mapped to services, then how services are mapped to resources does not affect nor concern intents; (2) intents become the first-class citizen, meaning that they appear at any level and therefore their intent can be a queryable characteristic. The first option is indeed what applies to the TOSCA standard and is currently covering the industrial scenarios we encountered in our practice. Nevertheless, option (1) does not preclude option (2).

In the future iterations of the TOSCA technical committee specification work, we plan to look further into the above options to tentatively extend the standard in supporting runtime intent models; these models would allow orchestrators to maintain a steady-state "version" of the intent-model such that operators can query and/or manipulate that version for a variety of purposes, e.g., elasticity, QoS guarantees or privacy-by-design, to name a few.

7 Conclusions and future work

This paper outlines and discusses intent modelling, an infrastructure-as-code modelling and specification approach that mixes several tenets from the declarative modelling school and elaborates on the "implementation" of intent modelling by means of the OASIS TOSCA standard.

Using simple and yet illustrative examples with connected tool-support, we outlined how the intent modelling approach offers quality of description and, as a consequence, quality universal orchestrations for modern cloud service applications.

In the future we plan to elaborate further on the foundations of intent modelling from a more theoretical and practical perspective, offering practitioners an overview and discussion on how intent modelling and the take that the TOSCA standard has on it can aid in the resolution of complex orchestration tasks such as infrastructure management. In so doing, we plan to semi-formalise our experiences in a full account from the trenches where TOSCA is being applied, to offer a practical war-report that orchestration practitioners can use for future references in their own practice.

References

1. Bass LJ, Weber IM (2015) DevOps - A Software Architect's Perspective., ser. SEI series in software engineering. Addison-Wesley, Boston
2. Peltz C (2003) Web services orchestration and choreography. Computer 36(10):46–52
3. Kavakli E (1999) Goal-driven requirements engineering: modelling and guidance. Ph.D. dissertation, Citeseer
4. Lipton P, Palma D, Rutkowski M, Tamburri DA (2018) Tosca solves big problems in the cloud and beyond! IEEE Cloud Comput. https://doi.org/10.1109/MCC.2018.111121612
5. Brunet J, Semmak F, Laleau R, Gnaho C (2008) Using variants in kaos goal modelling. In: ICEIS (3-2), J. Cordeiro and J. Filipe, Eds., pp. 339–344, 978-989-8111-38-8
6. Binz T, Breiter G, Leymann F, Spatzier T (2012) Portable cloud services using tosca. IEEE Internet Comput 16(3):80–85. [Online]. Available: http://dblp.uni-trier.de/db/journals/internet/internet16.html#BinzBLS12
7. Breiter G, Behrendt M, Gupta M, Moser S, Schulze R, Sippli I, Spatzier T (2014) Software defined environments based on tosca in IBM cloud implementations. IBM J Res Dev 58(2):1–10
8. Brogi A, Soldani J (2016) Finding available services in tosca-compliant clouds. Sci Comput Program 115–116:177–198. [Online]. Available: http://dblp.uni-trier.de/db/journals/scp/scp115.html#BrogiS16
9. Martino BD, Cretella G, Esposito A (2017) A comparison between tosca and openstack hot through cloud patterns composition. IJGUC 8(4):299–311. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijguc/ijguc8.html#MartinoCE17
10. Goedertier S, Vanthienen J, Caron F (2015) Declarative business process modelling: principles and modelling languages. Enterp IS 9(2):161–185

11. Pichler P, Weber B, Zugal S, Pinggera J, Mendling J, Reijers H (2012) In: Business process management workshops, ser. lecture notes in business information processing. In: Daniel F, Barkaoui K, Dustdar S (eds) Imperative versus declarative process modeling languages: An empirical investigation. Springer, Berlin, pp 383–394
12. Van Lamsweerde A (2009) Requirements engineering: from system goals to UML models to software specifications. Wiley Publishing, Hoboken
13. Ernst NA, Yu Y, Mylopoulos J (2006) Visualizing non-functional requirements. In: 2006 first international workshop on requirements engineering visualization (REV'06-RE'06 Workshop). IEEE, pp. 2–2
14. Santos E, Nguyen H (2009) Enabling a collaborative problem-solving framework through user intent modeling of the analytic process. DTIC Document, Technical Report
15. Bratman ME (1990) Intention, plans and practical reason. Harvard University Press, Cambridge
16. Gall NR (2000) John d. greenwood, ed., the future of folk psychology: Intentionality and cognitive science; scott m. christensen and dale r. turner, eds., folk psychology and the philosophy of mind. Minds Mach 10(3):416–423
17. Malle BF, Moses LJ, Baldwin DA (2001) Intentions and intentionality: foundations of social cognition. MIT press, Cambridge
18. Courbis C, Finkelstein A (2005) Weaving aspects into web service orchestrations. In: ICWS '05: Proceedings of the IEEE international conference on web services (ICWS'05). Washington, DC, USA: IEEE computer society, pp 219–226
19. Nitto ED, Jamshidi P, Guerriero M, Spais I, Tamburri DA (2016) A software architecture framework for quality-aware devops. D. Ardagna, G. Casale, A. van Hoorn, and F. Willnecker, Eds. ACM, pp. 12–17
20. OMG (2011) Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03. www.omg.org
21. Kuang L, Wu J, Deng S, Li Y, Wu Z (2006) Service classification using adaptive back-propagation neural network and semantic similarity. In: CSCWD. IEEE, pp. 834–838
22. TOSCA, "TOSCA Simple Profile in YAML Version 1.0," OASIS Standardization Board formal/2016-08-17, Tech. Rep., 2016. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf
23. Lipton P, Palma D, Rutkowski M, Tamburri DA (2018) TOSCA solves big problems in the cloud and beyond! IEEE cloud computing 5:37–47
24. Lamsweerde AV (2001) Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE international symposium on requirements engineering, vol. 249. Toronto, Canada, p. 263
25. Regev G, Wegmann A (2005) Where do goals come from: the underlying principles of goal-oriented requirements engineering. In: RE '05: Proceedings of the 13th IEEE international conference on requirements engineering (RE'05). IEEE Computer Society, Washington, DC, USA, pp 253–362

Damian A. Tamburri is an Assistant Professor at the Technical University of Eindhoven and the Jheronimus Academy of Data Science in s'Hertogenbosch, The Netherlands. He serves as active voting member of the TOSCA Technical Committee. His current research interests lie mainly in advanced software architecture styles (e.g., SOA, Big-Data, etc.), advanced software architecting methods (e.g., MDA, continuous architecting and DevOps), and social software engineering (Socio-technical congruence, Measuring Social Debt, etc.) and their investigation by means of Empirical Software Engineering. Contact him at d.a.tamburri@tue.nl.

Willem-Jan Van Den Heuvel is a full professor in Information Systems and managing director of the European Research Institute of Services Science (ERISS). He is currently the scientific director of several NL and H2020 projects including the recently funded H2020 ANITA project focusing on evolutionary and collaborative software technology for digital and cyber crime-fighting. His research interests are at the cross-junction of software service systems and business process management with an emphasis on (global) networked enterprises. In particular, his expertise revolves around the following major research themes: business process management, Big data analytics, software service engineering (including service governance) and software legacy services modernization. In RADON, he will act as Scientific coordinator.

Chris Lauwers is the Founder and Chief Executive Officer of Ubicity Corporation. Chris has 25 years of experience designing and building networking and communications solutions for large enterprises and service providers. Prior to Ubicity, Chris served as Chief Technology Officer of Avistar Communications Corp from 2001 to 2012. He was also Avistar's Vice President of Engineering from 1996 to 2001 and Director of Engineering from 1994 to 1996. Before Avistar, he served as Principal Software Architect at Vicor, Inc. from 1990 to 1994 and as a Research Associate at Olivetti Research Center from 1987 to 1990. Chris holds a B.S. in Electrical Engineering from the KU Leuven of Belgium and M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from Stanford University. He holds 91 patents in the areas of presence-based interactions, desktop video, recorded and live media at the desktop, multimedia documents, and data sharing.

Paul Lipton is VP of Industry Standards and Open Source at CA Technologies where he coordinates strategy and participation in these areas. He co-chairs the TOSCA Technical Committee, and also serves on the Board of Directors of OASIS, the Eclipse Foundation, the OMG, and the DMTF. He is an approved US delegate to the SO/IEC JTC 1 initiatives focused on cloud standards, Internet of Things, and Smart Cities. Contact him at Paul.Lipton@ca.com.

Derek Palma is Founder and CTO at Vnomic, a pioneer in the declarative delivery and governance of complex applications on software defined infrastructures. He is an expert in model based systems and desired-state automation, co-leader of the TOSCA Instance Model group, and OASIS TOSCA co-editor. Contact him at dpalma@vnomic.com.

Matt Rutkowski is Senior Engineer and Master Inventor at IBM and has worked to develop open infrastructure and industry standards and open source for over 15 years in areas such as Government, Banking and Digital Media and Entertainment. Most recently, he has been working on cloud, serverless technology, and software security standards. He was lead editor for the OASIS IDCloud TC, founder and Co-Chair of the DMTF Cloud Auditing Work Group and also is Co-Chair of the OASIS TOSCA Interop Subcommittee and a Technical Committee contributor/editor. Contact him at mrutkows@us.ibm.com.