Workflows in TOSCA vs Similar DSLs

An In-Depth Comparison Between Model-Driven and Task-Driven Workflows

Introduction

Let's all agree that pretty much no production deployment, and/or orchestration of the deployed infrastructure and VNFs, can truly be automated without workflows. Have you asked yourself how workflows are handled in TOSCA? I have. And what I found is no less complex and challenging than the overall TOSCA landscape.

Workflows appeared in TOSCA 1.0 (XML) but were not included in TOSCA Simple Profile for YAML 1.0. They are making a comeback, however, in the yet-to-be-approved TOSCA Simple Profile for YAML 1.1, and may continue to evolve beyond this.

So, how were workflows handled by the TOSCA users in the time period from 2013 (when TOSCA 1.0 first emerged) until now - or, more correctly - the time when TOSCA Simple Profile for YAML (Yet Another Markup Language) will be published?

Companies and open source projects apparently filled the four-year gap by continuing with their own workflows, which allowed industry to progress, and possibly put pressure on OASIS TOSCA to refocus on workflows in 2017.

In the meantime, open source implementations such as Cloudify, and more recently ARIA, are already supporting workflows via TOSCA-like Domain Specific Languages (DSLs). These are DSLs that use YAML as the markup language and have been inspired by TOSCA 1.0 (XML), but in many ways have leapfrogged the TOSCA Simple Profile for YAML, workflows and more.

The image below is a snapshot of timelines showing the TOSCA specifications versus those DSLs, and it is useful to understand why those DSLs may still be very much needed in 2017 until the maturation of TOSCA.

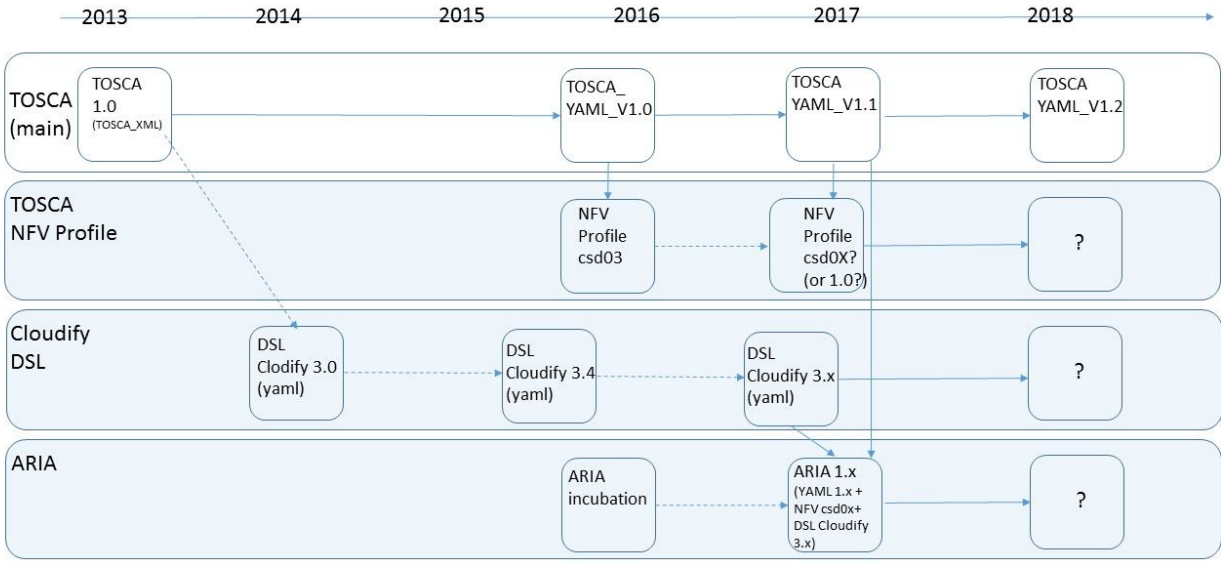


Figure 1: TOSCA/DSL landscape (viewed through the "workflows" perspective)

And then, of course, there are broader and more complex languages focusing on business process modelling (BPMN/BPML) and business process execution (BPEL) that preceded the TOSCA handling of workflows. They were, in fact, developed before the acceleration of the move to cloud, so from a DevOps perspective they can be considered "legacy".

Workflow Defined

This is how we will define workflow. A workflow is an orchestrated and repeatable pattern representing an execution plan expressed as a directed graph of tasks that is simple to define, reason, execute, and visualize.

DevOps workflows are used to orchestrate operations in infrastructure and applications, automate complex CI/CD processes etc. A typical use case for a DevOps workflow includes "Day 0" operations such as an install/uninstall workflow which executes the lifecycle operation of an application and its associated infrastructure.

Things get vastly more complex when a workflow deals with processes that need to update/change an existing deployment, based on "Day 2" operations. Typical workflows that fit into this category include:

- Heal workflow - fix a failed node (typically by re-executing the failed node lifecycle operation and relationship)
- Scale workflow - increase the capacity of a given application node (typically by executing the lifecycle operation of the target node multiple times on each scale instance)
- Update workflow - a common pattern for update is referred to as blue/green. (typical

implementation includes taking a snapshot of the current state -> pushing the new update -> redirecting traffic to the new deployment -> gradually redirecting all traffic to the new version -> if failed, roll-back to the previous version)

- Topology update (aka blueprint update) - adding/removing an entire service from and existing application topology. A typical implementation includes comparing the new blueprint with the existing version -> identifying the changes -> executing the changes and updating the model accordingly.

Task-Based vs Model-Based Workflows

There are basically two main approaches to workflow execution: a task-based approach and a model-driven approach. In a task-based system we execute a workflow by breaking it into a specific set of tasks, mapping each step, and executing it. For example, let's imagine a simple use case of installing a nodeJS instance on a virtual machine. The workflow for that task includes the following steps:

1. Provision the VM
2. Setup network (public/private IP, security group, etc)
3. Install and configure the nodeJS instance
4. Start the nodeJS instance

In a model-driven system I would define the model first, and the workflow execution is driven from the model and generated implicitly. The core components of a model-driven workflow are:

- Model (Graph) - describes the nodes and their relationships
- State - maintain the actual state of execution of each node
- Execution - the actual code that maps to the specific lifecycle operation of each node

The key difference between the model-driven and task-driven approach is that task-driven workflows tend to be purpose-built (allows for more specificity, but less re-usability) while the model-driven approach is more generic and flexible.

In other words, a model-driven workflow is simpler to change as it groups the tasks into their respective node-types and thus provides more building blocks that are easier to change and maintain. It's also easier to read and follow, and fits better with a designer's task.

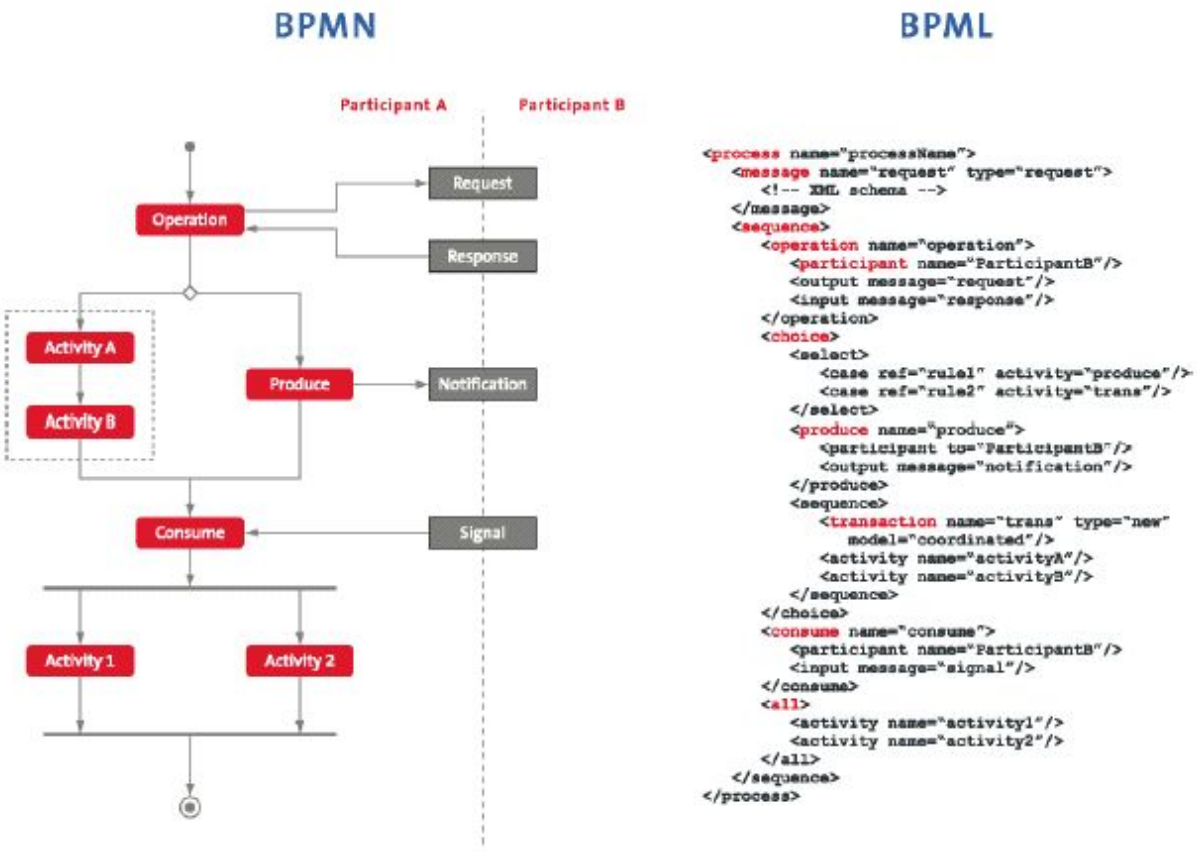
Business Process Modeling In More Detail

With this in mind, let's take a look at some of the languages mentioned before.

BPMN/BPML

Business Process Modeling Notation (BPMN) is what a business process analyst will use to design executable business processes. BPMN is used in the design phase, and is not an executable language. BPMN directly translates into BMPL that is then executed using an execution engine. Business Process Modeling Language (BPML) is a "meta-language for the modeling of business processes."[1] It is compared to XML which is a meta-language for the modeling of business data. BPML provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional, finite-state machine.

The figure below shows an example of how BPMN and BPML relate to each other.

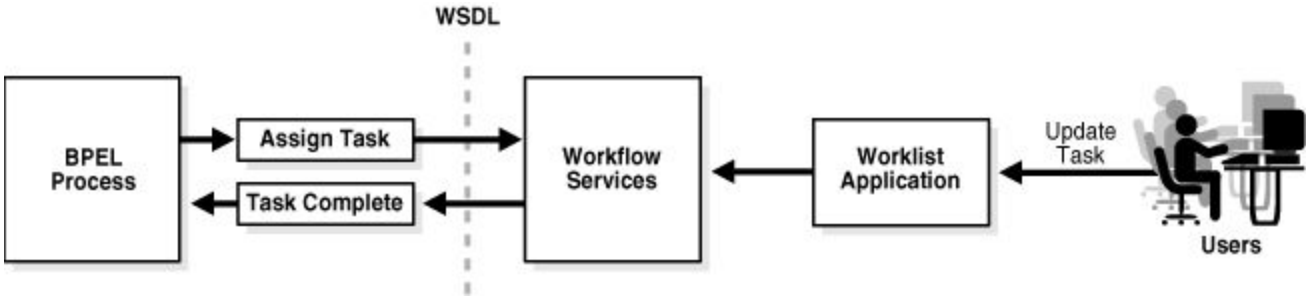


[1] bpmi.org

BPEL

Next came BPEL, which some consider a super-set, others a subset, of BPML, but it is the de-facto BPMN SOA/Web pairing as an execution language. Web services by themselves just perform atomic actions. They take in a group of inputs and provide some output. In order to have Web Services cooperate to provide a greater degree of service, there needs to be a means of specifying the interaction of cooperating services. BPEL4WS provides this means. It specifies the order in which the web services will be invoked, provides a mechanism for recovering from faults, and provides consistency and reliability for Web service applications. BPEL4WS is based on XML. A business process, adequately described in BPEL, is executable by a BPEL engine, just like a process modeled in BPML is executable by a BPML engine.

Workflow services in BPEL enable you to interleave human interactions with connectivity to systems and services within an end-to-end process flow. As shown in the figure below, workflow services are linked to a BPEL process through a WSDL (Web Service Definition Language) contract, like any other Web service. The process assigns a task to a user or role and waits for a response.



What do BPML and BPEL have in common? First off, the conformance to BPMN. Second, XML. Third, they are both task-driven when it comes to workflows. In other words, they are geared towards a bottom-up approach where one assumes each task as a process (including the workflow task) and they exchange requests and responses as dictated by the tasks.

TOSCA XML

TOSCA_XML came about very much under the influence of SOA and XML, and defers to the BPML handling of workflows by introducing “plans” (runtime aspects are addressed by providing a container for specifying models of plans which support the management of instances of services).

Plans defined in a Service Template describe the management aspects of service instances, especially their creation and termination. These plans are defined as process models, i.e. a workflow of one or more steps. Instead of providing another language for defining process models, the specification relies on existing languages like BPMN or BPEL. Relying on existing standards in this space facilitates portability and interoperability, but any language for defining process models can be used. The TOSCA metamodel provides containers to either refer to a

process model (via Plan Model Reference) or to include the actual model in the plan (via Plan Model). A process model can contain tasks (using BPMN terminology) that refer to operations of Interfaces of Node Templates (or operations defined by the Node Types specified in the type attribute of the Node Templates, respectively), operations of Interfaces of Relationship Templates (or operations defined by the Relationship Types specified in the type attribute of the Relationship Templates, respectively), or any other interface (such as the invocation of an external service for licensing). In doing so, a plan can directly manipulate nodes of the topology of a service or interact with external systems.

Making a concrete instance of a Topology Template can be done by running a corresponding Plan (so-called instantiating management plan, aka build plan). This build plan could be provided by the service developer who also creates the Service Template. The build plan can be adapted to the concrete environment of a particular service provider. Other management plans, useful in various states of the whole lifecycle of a service, could be specified as part of a Service Template. Similar to build plans, such management plans can be adapted to the concrete environment of a particular service provider.

Even with its partiality towards BPML, it is to be commended that TOSCA_XML principals had the wisdom to avoid adopting and incorporating a particular business process modelling specification, and instead chose to just have an optional mechanism by which the use of such a specification could be defined. Remember this point, since we will come back to it later.

XML? I Think Not

So it is 2013, TOSCA_XML arrives. However, it is loaded with XML, with the implication that it can potentially use BPML or BPEL for workflow, and no one is really ready to use it because the development community (DevOps in particular) hates XML.

What is the industry to do? The industry responded as it usually responds when a standard is created that is technically great, but does not jive well with the evolution of other technologies in the domain that it is supposed to be applied (in this case, cloud): it adopted the TOSCA philosophy, but moved away from XML towards YAML, from imperative task-oriented architecture towards declarative model-driven architecture, and from task-oriented workflows towards model-oriented workflows.

Cloudify and ARIA Circle Back to TOSCA

Cloudify's DSL is a good example of this trend. It started by following the initial TOSCA_XML philosophy, but avoided the "baggage" (XML, BPMN/BPML/BPEL), and has early on (ahead of the emergence of TOSCA_YAML_V1.0) embraced the use of YAML and a declarative, model-oriented workflow approach. With the inclusion of ARIA in 2017 it will come full circle and be fully compliant to TOSCA_YAML_V1.0 and future versions approved by OASIS. Until the maturation of TOSCA_YAML, ARIA will support both the latest approved TOSCA_YAML (now V1.0) and Cloudify DSL, allowing service providers to choose to use the one that satisfies their requirements better.

In Cloudify's DSL, workflows take the model-driven approach, and are effectively automation

process algorithms. They describe the flow of the automation by determining which tasks will be executed and when. A task may be an operation (implemented by a plugin), but it may also be other actions, including arbitrary code. Workflows are written in Python, using dedicated APIs and framework.

Workflows are deployment specific. Every deployment has its own set of workflows (declared in the blueprint – the equivalent of a TOSCA service template), and executions of a workflow are in the context of that deployment. Workflows can have parameters which are declared in the blueprint, and each parameter can be declared as either mandatory or optional with a default value.

Cloudify comes with a number of built-in workflows - currently these are the workflows for application install and uninstall, as well as a generic workflows for executing operations called execute_operation. It also supports the writing of custom workflows, and includes built-in workflows, the latter being declared and mapped in the blueprint in *types.yaml*, which is usually imported either directly or indirectly via other imports.

```
# snippet from types.yaml
workflows:
  install: default_workflows.cloudify.plugins.workflows.install
  uninstall: default_workflows.cloudify.plugins.workflows.uninstall
  execute_operation:
    mapping: default_workflows.cloudify.plugins.workflows.execute_operation
    parameters:
      operation: {}
      operation_kwargs:
        default: {}
      run_by_dependency_order:
        default: false
      type_names:
        default: []
      node_ids:
        default: []
      node_instance_ids:
        default: []
```

Built-in workflows are not special in any way - they use the same API and framework as any custom workflow is able to use, and one may replace them with different workflows with the

same names.

Controlling workflows (i.e. executing, cancelling, etc.) is done via REST calls to the management server. In this guide, the examples will be shown using Cloudify CLI commands which in turn call the above REST API calls.

TOSCA Moves to YAML and Declarative, Model-Oriented Workflows

I don't know this for a fact, but I would like to think that the work done in the industry (e.g. Cloudify DSL) defining TOSCA-like DSLs that leverage YAML and declarative, model-oriented workflows both inspired and challenged the OASIS TOSCA principals to re-focus their efforts.

As a result, TOSCA_YAML_V1.0 was approved towards the end of 2016. It moved away from XML and the use of BPMN/BPML/BPEL. Here is how that specification summarizes itself:

> *"The TOSCA language introduces a YAML grammar for describing service templates by means of Topology Templates and towards enablement of interaction with a TOSCA instance model perhaps by external APIs or plans. The primary currently is on design time aspects, i.e. the description of services to ensure their exchange between Cloud providers, TOSCA Orchestrators and tooling."*

This version of the specification has completely removed the TOSCA_XML "plans", and has not included an alternative for the handling of workflows (perhaps deliberately for its initial version).

However, perhaps an inadvertent leftover reminds us of the "special service plans", in the definition of the Instance Model:

> *"A deployed service is a running instance of a Service Template. More precisely, the instance is derived by instantiating the Topology Template of its Service Template, most often by running a special plan defined for the Service Template, often referred to as build plan."*

More remarkably, however, is the overall shift towards the declarative style. While the word "declarative" never appeared once in the TOSCA_XML, it appears in several sections in TOSCA_YAML_V1.0, for example in the TOSCA type definition, TOSCA Networking and TOSCA Policies sections. According to the specification:

> *"The declarative approach is heavily dependent of the definition of basic types that a declarative container must understand. The definition of these types must be very clear such that the operational semantics can be precisely followed by a declarative container to achieve the effects intended by the modeler of a topology in an interoperable manner."*

Furthermore, despite the absence of support for workflows in this first version, the intent regarding how to eventually handle workflows is evident in the description of Declarative Network Configuration:

> *"TOSCA orchestrators are responsible for the provisioning of the network*

connectivity for declarative TOSCA Service Templates (Declarative TOSCA Service Templates don't contain explicit plans). This means that the TOSCA orchestrator must be able to infer a suitable logical connectivity model from the Service Template and then decide how to provision the logical connectivity, referred to as "fulfillment", on the available underlying infrastructure."

Last, but not least, the TOSCA Simple Profile for NFV includes the notion of a "node state" (inexistent in TOSCA_XML) – another indication of the intent to move towards model-driven architecture and model-oriented workflows, away from task-oriented workflows:

> *"As components (i.e., nodes) of TOSCA applications are deployed, instantiated and orchestrated over their lifecycle using normative lifecycle operations (see section 5.7 for normative lifecycle definitions) it is important define normative values for communicating the states of these components normatively between orchestration and workflow engines and any managers of these applications"*. The following node states are now recognized: initial, creating, created, configuring, configured, starting, started, stopping, deleting, error – with additional states considered for future versions of the specification.

TOSCA Ready for Success With v1.1 and v1.2

Overall, the TOSCA DSL captured under the TOSCA_YAML_V1.0 represents great progress, in terms of alignment with industry and development community trends. Despite that, compliance to this specification is insufficient to address automation of deployment or orchestration of cloud workloads, and until topics such as workflow handling are addressed, TOSCA-like DSLs that are more mature, albeit proprietary, provide a much needed alternative.

I am looking forward to the approval of the TOSCA_YAML_V1.1 in 2017, and monitoring with interest the already on-going work on TOSCA_YAML_V1.2.

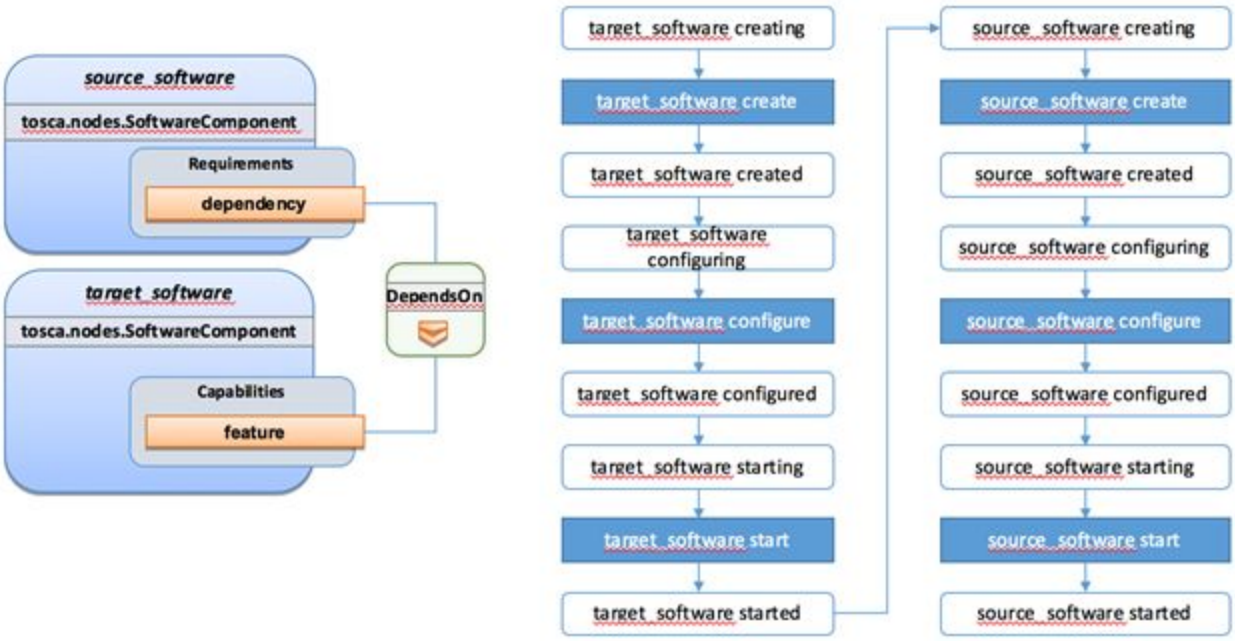
The new, still-to-be-approved V1.1 defines two different kinds of workflows that can be used to deploy (instantiate and start), manage at runtime or undeploy (stop and delete) a TOSCA topology: declarative and imperative workflows. Declarative workflows are automatically generated by the TOSCA orchestrator based on the nodes, relationships, and groups defined in the topology. Imperative workflows are specified by the author of the topology and allow the specification of any use-case that has not been planned in the definition of node and relationship types or for advanced use-cases (including reuse of existing scripts and workflows).

Workflows can be triggered on deployment of a topology (deploy workflow), undeployment (undeploy workflow), or during runtime, manually or automatically, based on policies defined for the topology.

Declarative workflows are the result of the weaving of a topology's nodes, relationships, and groups workflows. The weaving process generates the workflow of every single node in the topology, inserts operations from the relationships and groups, and finally, adds ordering consideration. The weaving process will also take care of the specific lifecycle of some nodes and the TOSCA orchestrator is responsible to trigger errors or warnings in case the weaving cannot be processed, or leads to cycles, for example.

When a node is abstract the orchestrator is responsible for providing a valid matching resources for the node in order to deploy the topology. The lifecycle of such nodes is the responsibility of the orchestrator and they may not answer to the normative TOSCA lifecycle. Their workflow is considered as "delegate" and acts as a black-box between the initial and started states in the install workflow and the started to deleted states in the uninstall workflow.

TOSCA relationships between nodes impacts the workflow generation to enable the composition of complex topologies. For example, the *dependsOn* relationship is used to establish a dependency from one node to another. A source node that depends on a target node will be created only after the other entity has been started – as in the figure below, where one custom software component depends on another one.



Imperative workflows are user defined and can define any specific constraints and ordering of activities. They are really flexible and powerful, and can be used for any complex use-case that cannot be solved with declarative workflows. However, they provide less reusability as they are defined for a specific topology rather than being dynamically generated based on the topology content.

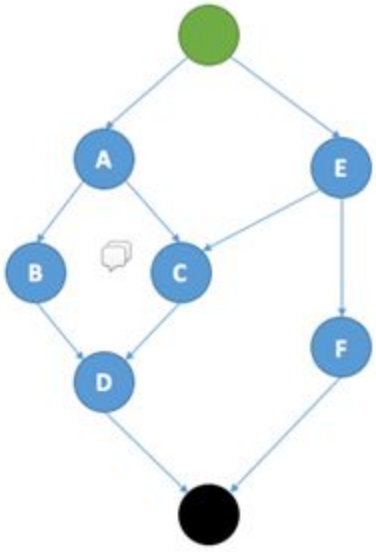
Imperative workflow grammar offers two ways to define a sequence of operations:

- Leverage the *on_success* definition to define the next steps that will be executed in parallel.
- Leverage a sequence of activity in a step.

The following example defines multiple steps and the *on_success* relationship between them.

```
topology_template:
  workflows:
    deploy:
      description: Workflow to deploy the application
      steps:
        A:
          on_success:
            - B
            - C
        B:
          on_success:
            - D
        C:
          on_success:
            - D
        D:
        E:
          on_success:
            - C
            - F
        F:
```

The following schema is the visualization of the above definition in term of sequencing of the steps.



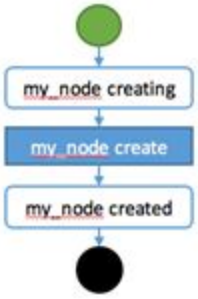
The step definition of a TOSCA imperative workflow allows multiple activities to be defined:

```
workflows:
 my_workflow:
  steps:
   create_my_node:
    target: my_node
    activities:
     - set_state: creating
     - call_operation: tosca.interfaces.node.lifecycle.Standard.create
     - set_state: created
```

The sequence shown here defines three different activities that will be performed in a sequential manner. This is just equivalent to writing multiple steps chained together by *on_success*:

```
workflows:
 my_workflow:
  steps:
   creating_my_node:
    target: my_node
    activities:
     - set_state: creating
    on_success: create_my_node
   create_my_node:
    target: my_node
    activities:
     - call_operation: tosca.interfaces.node.lifecycle.Standard.create
    on_success: created_my_node
   created_my_node:
    target: my_node
    activities:
     - set_state: created
```

In both situations the resulting workflow is a sequence of activities:



In conclusion, the turnaround in TOSCA towards the declarative, model-driven approach for workflows is a very positive move, and will help TOSCA have “a good year in 2017”. There is really nothing wrong with using BPMN/BPML/BPEL, where appropriate. But there is something to be said when we try to use a complex specification designed for a different set of applications, in a domain where an approach that is simple, flexible, and reusable fits better - and the word is “overkill”.

So let’s support the “good year of TOSCA” initiative by getting behind TOSCA_YAML_V1.1, and focus on the progress of the improved future versions (TOSCA_YAML_V1.2 and beyond)!

Abbreviations

The different standard specifications referred to in the above article are as follows:

TOSCA_XML: Topology and Orchestration Specification for Cloud Applications Version 1.0

TOSCA_YAML: TOSCA Simple Profile in YAML (followed by the version)

BPMN: Business Process Model and Notation

BPML: Business Process Modeling Language

BPEL: Business Process Execution Language

DSL: Domain Specific Language

WS: Web Services

SOA: Service Oriented Architecture

XML: eXtensible Markup Language

YAML: Yet Another Markup Language