Instance Model for TOSCA Version 1.0

Committee Specification Draft 01

16 November 2017

Specification URIs

This version:

http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/csd01/TOSCA-Instance-Model-v1.0-csd01.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/csd01/TOSCA-Instance-Model-v1.0-csd01.html
http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/csd01/TOSCA-Instance-Model-v1.0-csd01.docx

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.html
http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.docx

Technical Committee:

OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC

Chairs:

Paul Lipton (paul.lipton@ca.com), CA Technologies
John Crandall (jcrandal@brocade.com), Brocade Communications Systems

Editor:

Derek Palma (dpalma@vnomic.com), Vnomic

Related work:

This specification is related to:

- *Topology and Orchestration Specification for Cloud Applications Version 1.0*. Edited by Derek Palma and Thomas Spatzier. 25 November 2013. OASIS Standard. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

Abstract:

This specification defines the information model, behavioral semantics, and access methods for a dynamic object model representation of TOSCA Service Template deployments required to enable on-going automated management of TOSCA topologies. The Instance Model represents the current state of a deployment including all inputs, concrete node fulfillments, property settings, cardinalities, relationships, applied policies, and correlation with external resources and entities. Service template deployments can be queried, navigated, reasoned over and updated with imperative workflows, policy actions, and their evolution observed over time. The instance model is extensible, allowing integration of information from other domains, across deployments and orchestrators.

Status:

This document was last revised or approved by the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC on the above date. The level of approval is also

listed above. Check the “Latest version” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca#technical.

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list, after subscribing to it by following the instructions at the “Send A Comment” button on the TC’s web page at https://www.oasis-open.org/committees/tosca/.

This Committee Specification Draft is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (https://www.oasis-open.org/committees/tosca/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:

[TOSCA-Instance-Model-v1.0]

Instance Model for TOSCA Version 1.0. Edited by Derek Palma. 16 November 2017. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/csd01/TOSCA-Instance-Model-v1.0-csd01.html. Latest version: http://docs.oasis-open.org/tosca/TOSCA-Instance-Model/v1.0/TOSCA-Instance-Model-v1.0.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction

The TOSCA Domain Specific Language (DSL), expresses a topology of entities (service template), their lifecycles, management operations and constraints. Instantiation of a TOSCA service template involves an orchestrator processing a service template, fulfilling the requirements of each node template, and orchestrating a sequence of lifecycle operations across the entities of the topology to reach an end-state prescribed by the service template.

The TOSCA Instance Model expresses the complete structure and state of an instantiated service template. The existence of an instance model has been implied in other TOSCA specifications when describing the behavior semantics of a TOSCA orchestrator as it executes a service template deployment. A complete and concise specification of the Instance Model permits the continued automated management of an instantiated service template and is motivated via the following use cases:

1. Expressing the end-state reached of a TOSCA deployment by an orchestrator;
2. Expressing intermediate states as the deployment is changed by declarative or imperative orchestration;
3. Comparing a pair of TOSCA deployments, possibly by different orchestrators, for similarities or differences;
4. Enabling imperative workflows to examine and synchronize its knowledge of the state of the deployment;
5. Enabling imperative workflows to execute side-effects on the deployment;
6. Enabling declarative actions to be triggered as the deployment changes;
7. Expressing the state of the deployment over time, as entities in the topology change state due to their innate behaviors and external influences;
8. Correlating entities in the deployment to entities in the physical world.

A set of Instance Model operations are defined and specified to support the creation and dynamic evolution:

1. Derivation the Instance Model from a service template
2. Accessing the Instance Model
3. Updating the Instance Model
4. Correlation of the Instance Model entities with external entities
5. Synchronizing the Instance Model with external state

1.0 IPR Policy

This Committee Specification Draft is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/tosca/ipr.php).

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

1.3 Glossary

The following terms are used throughout this specification and have the following definitions when used in context of this document.

Term	Definition
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.
Node Template	A *Node Template* specifies the occurrence of a software component node as part of a Topology Template. Each Node Template refers to a Node Type that defines the semantics of the node (e.g., properties, attributes, requirements, capabilities, interfaces). Node Types are defined separately for reuse purposes.
Relationship Template	A *Relationship Template* specifies the occurrence of a relationship between nodes in a Topology Template. Each Relationship Template refers to a Relationship Type that defines the semantics relationship (e.g., properties, attributes, interfaces). Relationship Types are defined separately for reuse purposes.
Service Template	A *Service Template* is typically used to specify the "topology" (or structure) and "orchestration" (or invocation of management behavior) of IT services so that they can be provisioned and managed in accordance with constraints and policies. Specifically, TOSCA Service Templates optionally allow definitions of a TOSCA Topology Template, TOSCA types (e.g., Node, Relationship, Capability, Artifact), groupings, policies and constraints along with any input or output declarations.
Topology Model	The term Topology Model is often used synonymously with the term Topology Template with the use of "model" being prevalent when considering a Service Template's topology definition as an ***abstract representation*** of an application or service to facilitate understanding of its functional components and by eliminating unnecessary details.
Topology Template	A Topology Template defines the structure of a service in the context of a Service Template. A Topology Template consists of a set of Node Template and Relationship Template definitions that together define the topology model of a service as a (not necessarily connected) directed graph. The term Topology Template is often used synonymously with the term Topology Model. The distinction is that a topology template can be used to instantiate and orchestrate the model as a ***reusable pattern*** and includes all details necessary to accomplish it.

Abstract Node Template	An abstract node template is a node that doesn't define an implementation artifact for the create operation of the TOSCA lifecycle. The create operation can be delegated to the TOSCA Orchestrator. Being delegated an abstract node may not be able to execute user provided implementation artifacts for operations post create (configure, start etc.).
No-Op Node Template	A No-Op node template is a specific abstract node template that does not specify any implementation for any operation.

2 Summary of key TOSCA concepts

The TOSCA metamodel uses the concept of service templates to describe cloud workloads as a topology template, which is a graph of node templates modeling the components a workload is made up of and as relationship templates modeling the relations between those components. TOSCA further provides a type system of node types to describe the possible building blocks for constructing a service template, as well as relationship type to describe possible kinds of relations. Both node and relationship types may define lifecycle operations to implement the behavior an orchestration engine can invoke when instantiating a service template. For example, a node type for some software product might provide a 'create' operation to handle the creation of an instance of a component at runtime, or a 'start' or 'stop' operation to handle a start or stop event triggered by an orchestration engine. Those lifecycle operations are backed by implementation artifacts such as scripts or Chef recipes that implement the actual behavior.

An orchestration engine processing a TOSCA service template uses the mentioned lifecycle operations to instantiate single components at runtime, and it uses the relationship between components to derive the order of component instantiation. For example, during the instantiation of a two-tier application that includes a web application that depends on a database, an orchestration engine would first invoke the 'create' operation on the database component to install and configure the database, and it would then invoke the 'create' operation of the web application to install and configure the application (which includes configuration of the database connection).

The TOSCA simple profile assumes a number of base types (node types and relationship types) to be supported by each compliant environment such as a 'Compute' node type, a 'Network' node type or a generic 'Database' node type. Furthermore, it is envisioned that a large number of additional types for use in service templates will be defined by a community over time. Therefore, template authors in many cases will not have to define types themselves but can simply start writing service templates that use existing types. In addition, the simple profile will provide means for easily customizing existing types, for example by providing a customized 'create' script for some software.

3 TOSCA Instance Model Conceptual Overview

3.1 What is an instance model?

The Instance Model represents the current state of a deployment including all inputs, concrete node fulfillments, property settings, cardinalities, relationships, applied policies, and correlation with external resources and entities.

3.2 What problems does it address?

- Expressing the current state of a TOSCA deployment
- Correlating the nodes of an instance model with concepts, components, identities, resources and services outside of a TOSCA deployment
- Provides a concise description how Service Templates are transformed into instances clearly delineating aspects that are invariant and independent of the specific orchestrator and hosting infrastructure and the aspects which are variant or orchestrator specific
- Expressing intermediate states as the deployment is changed by declarative or imperative orchestration
- Comparing a pair of TOSCA deployments, possibly by different orchestrators, for similarities or differences
- Enabling imperative logic to examine and synchronize its knowledge of the state of the deployment
- Enabling imperative logic to execute side-effects on the deployment
- Enabling declarative actions to be triggered as the deployment changes
- Expressing the state of the deployment over time, as entities in the topology change state due to their innate behaviors and external influences
- Correlating entities in the deployment to entities in the physical world

4 Design Considerations

4.1 State-based Orchestration Paradigm

TOSCA uses a state-base orchestration paradigm where the orchestrator transitions the topology across a well-defined set of states to a set of final states.

4.2 Instance Model Consumers

This sections describes the actors which consume the instance model to gain knowledge of or cause changes to the TOSCA deployment.

4.2.1 Declarative Orchestration

The TOSCA orchestrator creates and updates the instance model while it is performing a declarative orchestration (an orchestration where it computes and executes all steps required to create a deployment topology which matches the topology template).

Constraints:

A. The TOSCA orchestrator is the only entity driving changes the to instance model during any (declarative or imperative) orchestration (changing means adding/removing node and relation instances)

4.2.2 Workflow Execution

Workflows may be executed by the TOSCA orchestrator to:

1. Delegate a subset of the topology orchestration to workflow to enable control beyond what is possible with declarative orchestration
2. Handle a policy action via invocation of a specific workflow

Constraints:

B. The TOSCA orchestrator triggers the execution of the workflow, during which it delegates responsibility to the workflow for transitioning topology nodes through their lifecycle. The TOSCA orchestrator updates the instance model based on state transitions and performs operation invocations specified in the workflow.

4.2.3 External Clients

External clients are entities which exist outside the implementation of the TOSCA orchestrator to:

1. Obtain dynamic information about the deployment topology including structure, node states, and node attribute values
2. Obtain static information about the topology including access to inputs, policies, templates, and types which fully describe the topology in TOSCA semantics

Constraints:

C. External clients can query the instance model directly but may see it changing across queries in terms of structure and state if an orchestration executes between queries.

D. External clients may see a time lag regarding the consistency of the instance model with the deployment topology structure and state. The change log may be queried to understand instance model evolution over time.
E. External client execution of workflows is arbitrated by the TOSCA orchestrator to ensure that all the orchestration execution constraints A and B are satisfied.

4.3 Orchestrator States vs. Real-world States

TOSCA orchestrators manage the state of nodes and transitions them from state to state[1]. This notion of state is somewhat artificial in that the orchestrator assumes a stable state is reached after an operation executes on a node without an error (currently an error results in an undefined state because there is nothing that specifies an error should result in the node returning to its previous stable state).This is because states denote the completion of lifecycle operations and lifecycle operations are fulfilled by node operation. Hence, state orchestration correlates to the invocation of lifecycle operations and not directly to the actual state of the node.

Additionally, orchestration states are only valid during orchestration. Once orchestration completes we must assume the states of nodes can change in various ways (failure, interactions with other nodes, user actions). Upon beginning orchestration, the orchestrator or the imperative workflow must decide the current state of all nodes in the topology and be willing to transition them to other states. There is the case where a state transition is not necessary, such as changing an operational configuration parameter that does not impact service integrity.

4.4 Reflecting Node Status in the Instance Model

The TOSCA instance model allows representing dynamic node information in node attributes. Best practices should be followed that focus on aggregating and summarizing information which is useful for managing the topology versus trying to create nodes which provide a very granular or timely representation of the state of physical nodes. It's critical that TOSCA implementations be able to provide scalable automation across large topologies and diverse combinations of node types.

For example, it is not practical to implement node types which offer their specific metric values as attributes since this places timeliness and consistency requirements for such information across the topology which can only be satisfied by appropriately designed monitoring systems. However, it is practical and useful to provide node status information which allows the orchestrator, using (declarative and imperative) workflows and policies to efficiently automate the management of nodes in terms of their dynamic behaviors.

4.5 Tracking State Changes

As orchestration progresses it is useful to fire events for each state transition across the topology. This event stream can be maintained for the life of a deployment. By maintaining a change log with events also denoting each orchestration beginning and end, clients can sync from a specific time without having to scan the entire topology.

4.6 Visibility of Nodes and Node Attributes

As nodes are transitioned through their states, a subset of attributes and relationships may be defined. For example, an automatically defined IP address may be defined after the server is provisioned as part of create but other attributes, such as additional interfaces and IPs, may not be defined until configured. Furthermore, this implies certain features of nodes may not be available until later lifecycle states (a

[1] We use the term "stable state" to mean the lifecycle states a node can end in. This is to differentiate from the "transition states" such as creating, installing, etc.

primary use case which TOSCA is designed to handle). This requires that in general TOSCA implies semantics such that not all attributes would be available in a given state.

We need to consider the following restrictions:

1. Nodes are only visible when they have a state defined (i.e. the orchestrator is dealing with their lifecycle)
2. Node attributes are only defined for the stable states
3. Node relationships are always navigable when the source and target nodes exist.
4. Orchestration state is never updated outside an orchestration. This is not even possible since TOSCA provides no way to propagate state changes from the node to the orchestrator and nodes don't have a state attribute.
5. Nodes can update their attributes (in implementation specific ways) with no specific guarantees in terms of precision or accuracy (i.e., it's a function of the node implementation). Of course, the attributes must be meaningful and timely to be of use.
6. As mentioned above, orchestration must somehow sync with the states of existing nodes regardless of declarative or imperative, although declarative can build the update sequence on the fly based on current states it can deduce.

4.7 Orchestration Execution Status

It is useful for the client to know if there is an active orchestration. This could be gotten from the change log or just by some status indicator describing the state of the topology. Most orchestrators offer this via their client APIs. This is currently out of the scope of TOSCA but useful to be able to assume this exists.

5 Model Structure

5.1 Meta Levels

A TOSCA description entails two levels of information: types and instances. Types help categorize things according to their characteristics or properties. For instance, all instances of the “tosca.nodes.Compute” type offer a public and a private address, a set of capabilities, and some requirements. By contrast, instances are the concrete things that belongs to these types (i.e., these categories) and they expose a value for each property their type defines. A given virtual machine, provisioned on Amazon EC2, whose addresses are known, is an instance of the type “tosca.nodes.Compute”.

TOSCA also supports the notion of template. A template acts as a predefined aggregate of instances, possibly incomplete as some values may be left undefined. We see templates as special constructors of instance aggregates (in the OO programming sense). One must provide these missing values to obtain a valid instance. Templates facilitate the construction of complex graphs of instances.

Figure 1 depicts how types, instances and templates relate one another. Templates set aside, the distinction between types and instances in TOSCA resemble the one between classes and objects in object-oriented programming, or the one between grammar rules and program instructions in compilers. In Figure 1 below, the instance model describes the blue box, that is the instances and the runtime information that characterize them, but it refers to TOSCA types and templates, which are described in separate documents (typically).

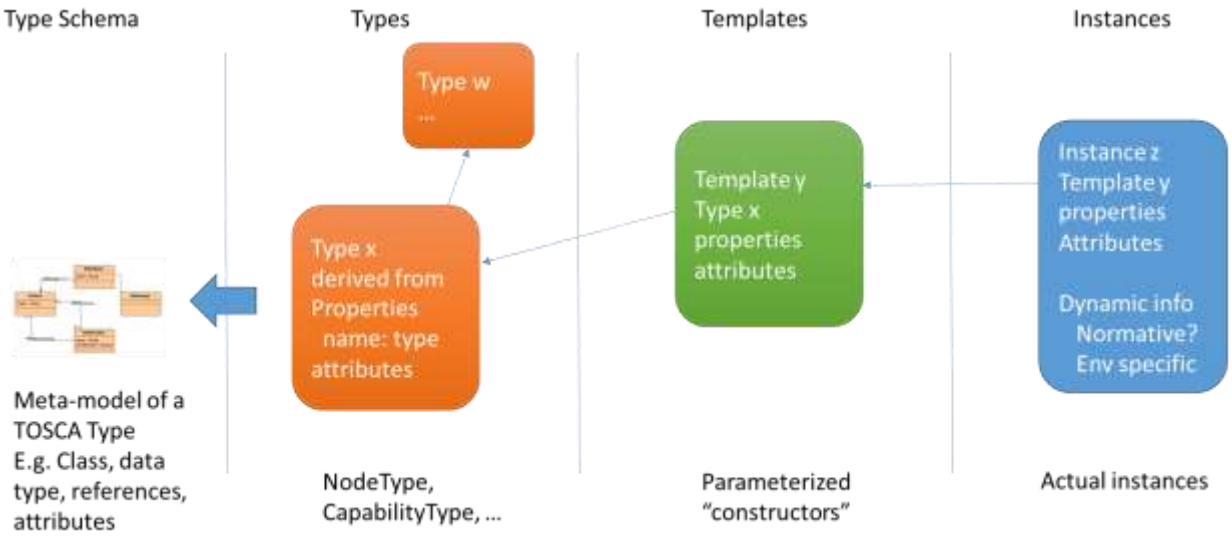


Figure 1 Information levels in TOSCA

Note the distinction between the “instance model” itself and its YAML representation. By instance model, we denote an abstract data structure (i.e., a graph of objects) that one can serialize in various formats, including XML, JSON or YAML, to name a few.

Meta-model == types

Model contains templates and instances.

Templates are a special instance constructor syntax. The select explicit domain values for properties.

Notes:

1. This is independent of the DSL and YAML (which encodes the DSL)

2. The DSL can express type and templates. It is able to express specific instance using the DSL, e.g. a set of instances for which the template intension syntax is not sufficient, i.e. supporting an extensional syntax.

5.2 Deriving the Instance Model

The instance model is a standard format for the information that any orchestrator must maintain at runtime. This information results from the TOSCA service template the orchestrator instantiates, including any relevant policies, provisioning choices made by the orchestrator, and monitoring data such as workload or performance metrics.

- Service template instantiation
  - Templates represent the entities that can be created
    - But cardinality may vary
    - Orchestrator will compute/know the cardinality at some point
  - Processing the templates
    - Apply inputs
    - All aspects of the service template must be resolved (substitutable, sub-topologies, …)
    - Policies matched and instantiated
  - The context used from the runtime environment should be reflected
- Additional entities provided by environment might also appear
  - Load balancer, Backup, Directory service, Monitoring, DNS, NTP endpoints …

Location/placement of container entities and resources

5.3 Sample Service Template

The following simple service template will be used for the derivations. More complex and sophisticated snippets will be incorporated as needed.

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a two-tier application on two servers.
(section 2-6 of YAML spec)

topology_template:
  inputs:
    # omitted here for brevity

  node_templates:
    wordpress:
      type: tosca.nodes.WebApplication.WordPress
      properties:
        # omitted here for brevity
      requirements:
        - host: apache
        - database_endpoint:
```

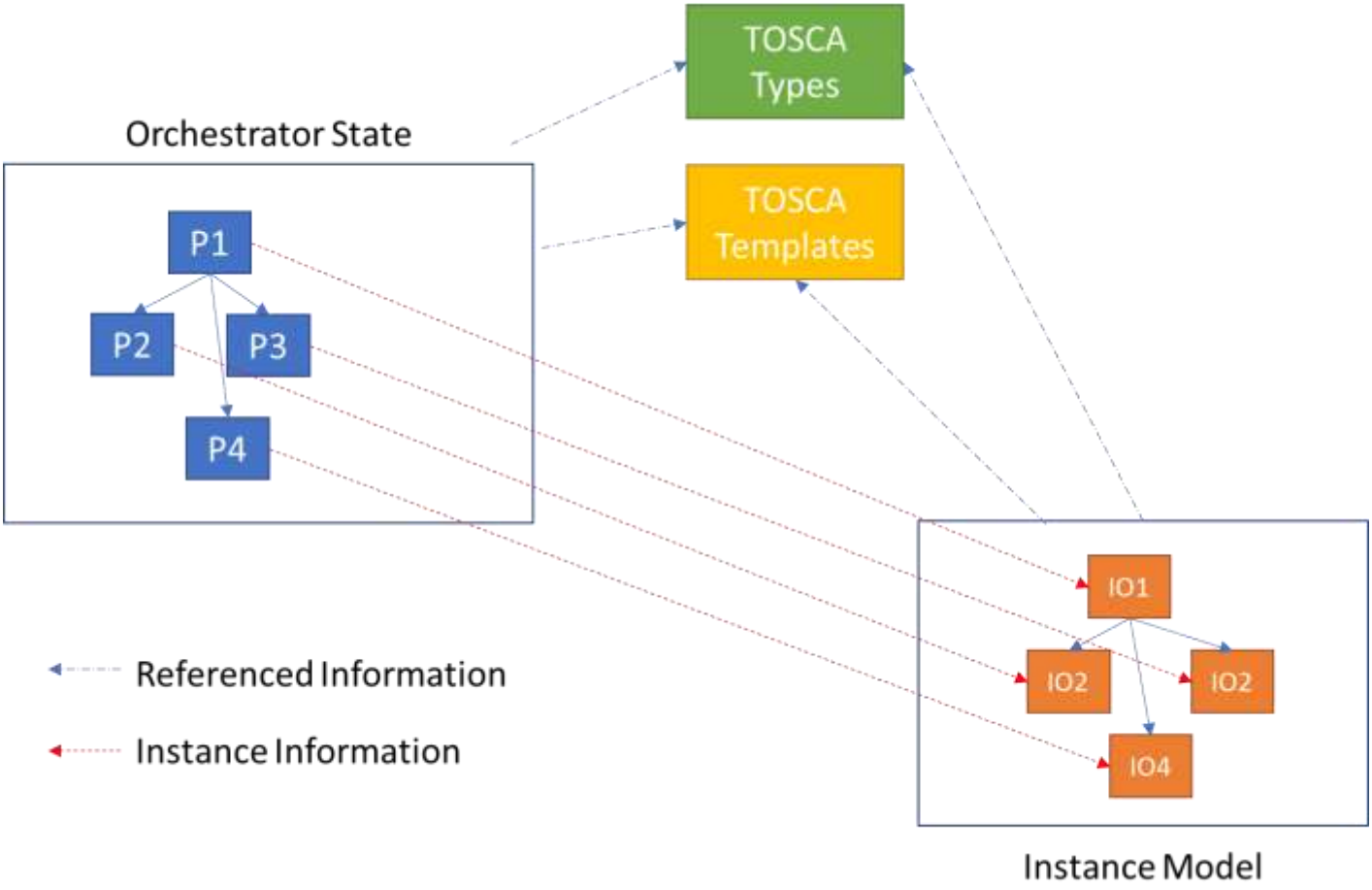


```
          node: wordpress_db
          relationship: my_custom_database_connection

  wordpress_db:
    type: tosca.nodes.Database.MySQL
    properties:
      # omitted here for the brevity
    requirements:
      - host: mysql

relationship_templates:
  my_custom_database_connection:
    type: ConnectsTo
    interfaces:
      Configure:
        pre_configure_source: scripts/wp_db_configure.sh

 # other resources not shown for this example ...
```

5.4 Deriving the Instance Model Graph

The orchestrator processes the service template, in dependency order, instantiating the correct number of instances for each node and transitioning the instances through their respective lifecycles. Node instances are created by the orchestrator using the *instance from node template derivation*, encoded in a local (orchestrator specific, not necessarily YAML or objects which directly represent YAML) representation. The result is a topology graph consisting of node instances as the nodes with the appropriate edges between pairs of node instances corresponding to the relationships defined in the service template, i.e., the instance model is explicit in terms of expressing relationships between nodes instances.

Node instances may be created by the orchestrator (or an API client via some external logic) without a node template using the *instance from node type derivation*. This supports the use case where fulfillment needs to pull other nodes into the topology to support the requirements of the service template without explicitly describing the nodes via a node template.



In general, (non-native TOSCA) orchestrators maintain their own proprietary (Ps) state representation of a service template deployment in their native data structures, which contain all the information required to derive the TOSCA instance model.

The instance model is a self-contained (and standalone) object model with *InstanceClasses* derived from the TOSCA DSL schemas and *InstanceObjects* derived from the proprietary orchestrator state.

A TOSCA orchestrator must maintain enough information to be able to constitute the contents of an instance model.

5.5 Instance Model Schema

5.5.1 Schema Modeling Language

We define the language we will use to express the instance model schema in Appendix C. This modeling language will be used throughout this document to encode the concepts in the instance model. Any analogous language can be used and thus can be mapped to an implementations specific language. The primary objective here is to provide a concise method for expressing the instance model by leveraging a simple and concise schema modeling language.

5.5.2 Instance Model Schema

This section defines the schema of the data in the instance model. This schema does not attempt to describe TOSCA DSL schema or replicate the TOSCA DSL metamodel. The instance model schema only needs to be able to represent a basic object model with containment hierarchies and cross references.

5.5.2.1 InstanceClass Definition

Instance class is the common schema of all instance model objects.

5.5.2.1.1 Data Types

Instance model schema modeling language supports the data types defined in the TOSCA DSL.

5.5.2.1.2 References

This section describes the specific kinds of references used in the instance model schema.

5.5.2.1.2.1 InstanceReference

InstanceReference relates a referencing instance model entity to a target instance model entity.

InstanceReferences use a path which navigates from a relative or root of the local instance model document. InstanceReference can reference instance model entities in other documents by prepending a URI which denotes the location of the document.

InstanceRefernce is based on JSON schema references, e.g.,

```
{"$ref": "#/definitions/person"}
```

is a local reference (references an item in the current document). A relative cross document reference would look like:

```
{"$ref": "../models/mymodel#/definitions/person"}
```

An absolute cross reference, such as into a model held in a repository, would look like:

```
{"$ref": "https://repo/models/othermodel#/definitions/person"}
```

The YAML form can be encoded with a similar key/value pair.

5.5.2.1.2.2 ExternalTosca<DocumentElement>Reference

ExternalToscaReference enables instance model entities to reference information expressed in the TOSCA DSL. The first part of the URI denotes the specific TOSCA DSL document, which may be kept in an orchestrator specific location. ExternalToscaReference reference may reference parts of an entity such as a node type's property.

An XPATH-like path denotes the path from the root of the target TOSCA document to the target TOSCA DSL element.

```
{"$externalToscaRef": "[http://]<tosca-
document>#/node_types/tosca.nodes.Root/attributes/tosca-id/type"}
```

Note that it may be desirable to reference invariant entities such as types without using a complete path, but using a logical name, such as the node type name. This will be explored and utilized to enable TOSCA DSL references without having to know the specific document the orchestrator "thinks" the TOSCA DSL entity is defined in.

```
{"$dsl-node-types": " tosca.nodes.Root"}
```

5.5.2.1.2.3 Reference scope and Semantics

Containment

Cross-reference

Local within instance model

Other documents

External to TOSCA DSL

5.5.3 InstanceObject Definition

Every object in the instance model extends from instanceObject. Instance Objects have attributes, references and operations. InstanceObject enables an object hierarchy representation of the instance model.

5.6 Instance Model Definition

Name	Type	Cardinality	Description
metadata	InstanceMetadata	1..*	Metadata describing the instance model. At least one entry describing the normal version of the schema the instance model complies with
nodes	List of root Instance Node	0..N	Contains the InstanceNode hierarchy
groups	List of root Instance Groups	0..N	Contains the InstanceGroup entities which define the InstanceNode membership of groups
inputs	List of InstanceInput	0..N	The values of all inputs used by the orchestrator
outputs	List of InstanceOutput	0..N	The values of all output resulting from the orchestration
policies	List of InstancePolicy	0..N	The policies applicable to entities in deployment

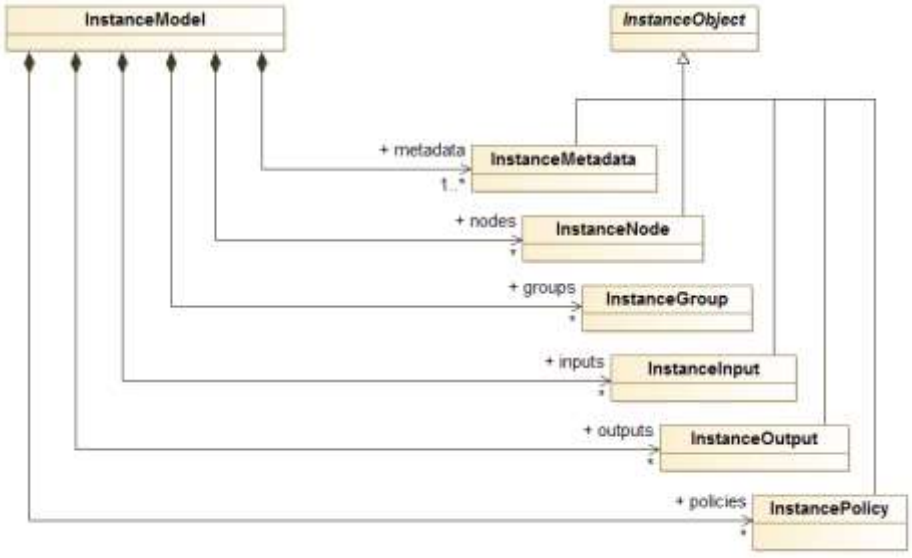


Figure 2 Overall structure of an instance model. The node instances, input and output instance and metadata instance.

5.6.1 InstanceNode Definition

The InstanceNode represents an instance of a node type in the instance model. Because it represents singular unique entity, all properties and capabilities values are defined (i.e., all ranges, optional values, etc. are defined).

Name	Type	Cardinality	Description
template	ExternalToscaReference	0..1 mutually exclusive with type	Allows navigation to the template used to create the instance.
type	ExternalToscaReference	0..1 mutually exclusive with template	Used when the node instance was not created with a template.
properties	InstanceProperty	0..N	Corresponds to the properties defined in the node type and desired values specified in the node template. Each property value indicates the final value used by the orchestrator for that property
attributes	InstanceAttribute	0..N	Corresponds to the attributes defined in the node type and holds the actual attribute values. Each attribute value indicates the value at the time the attribute was accessed. Attributes reflect the state of the underlying node and may change at any time.
capabilities	InstanceCapability	0..N	Corresponds to the capabilities defined in the node type and desired values specified in the node template. Each capability property value indicates the final value used by the orchestrator for that property
requirements	InstanceRequirement	0..N	Maps the source instance in a requirement relationship to the target instances. The mapping is 1 to N and does not have to be 1-1 or onto with respect to all source and target instances.

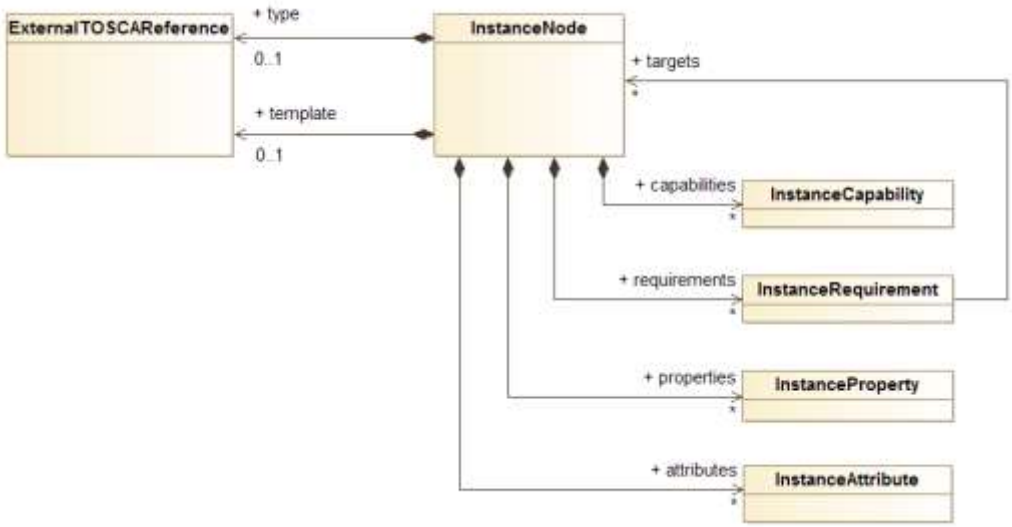


Figure 3 Structure of an InstanceNode, depicted as a UML class diagram

5.6.2 InstanceProperty Definition

For each property in the respective node type, a property with the same name is present in the InstanceNode with value set to the value the orchestrator represents as the current value of the property.

Each InstanceProperty is a <name, value> pair consisting of the name of the property in the respective TOSCA type and the value of the property encoded for the respective type of the property in the TOSCA type.

5.6.3 InstanceAttribute Definition

For each attribute in the respective node type, an attribute with the same name is present in the InstanceNode with value set to the value the orchestrator takes to be the current value of the attribute.

Attributes reflect the state of the underlying node and may change at any time. TOSCA implementations may vary significantly in their ability to reflect attribute changes at the rate at which the actual node instances change their attributes. Attributes are best thought of as a view into the state of a node instance and not necessarily a direct representation.

Each InstanceAttribute is a <name, value> consisting of the name of the attribute in the respective node type and the value of the attribute encoded for the respective type of the attribute in the node type.

5.6.4 InstanceCapability Definition

For each capability in the respective node type, a capability with this same name and schema as defined in the node type, where each property and attribute in this capability schema with the same name is present in the InstanceCapability with value set to the value the orchestrator takes to be the current value of the capability property.

Name	Type	Cardinality	Description
name	String	1	Indicates the name of the capability in the respective node type of the InstanceNode
properties	InstanceProperty	0..N	Corresponds to the properties defined in the node type and desired values specified in the node template. Each property value indicates the final value used by the orchestrator for that property
attributes	InstanceAttribute	0..N	Corresponds to the attributes defined in the node type and holds the actual attribute values. Each attribute value indicates the value at the time the attribute was accessed. Attributes reflect the state of the underlying node and may change at any time.

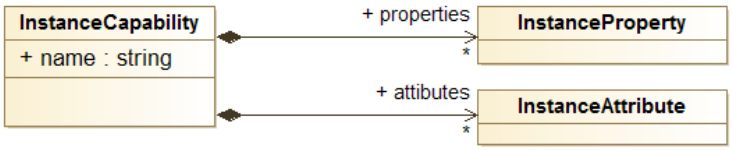


Figure 4 The structure of InstanceCapability, portrayed as a UML class diagram

5.6.5 InstanceRequirement Definition

For each requirement in the respective node type, a collection of references is created under the name of the requirement, each targeting the appropriate InstanceNodes in the topology graph.

Name	Type	Cardinality	Description
name	string	1	Indicates the name of the capability in the respective node type of the InstanceNode
targets	InstanceReference	0..N	Indicates the InstanceNodes fulfilling the requirement

Figure 5 The structure of InstanceRequirement, depicted as a UML class diagram

5.6.6 InstanceGroup Definition

The InstanceGroup represents an instance of a Group Type in the instance model.

Name	Type	Cardinality	Description
template	ExternalToscaReference	0..1 mutually exclusive with type	Allows navigation to the template used to create the instance.
type	ExternalToscaReference	0..1 mutually exclusive with template	Used when the node instance was not created with a template.
members	InstanceReference	0..N	References the InstanceNode members of the group.
properties	InstanceProperty	0..N	Corresponds to the properties defined in the group type and desired values specified in the group template. Each property value indicates the final value used by the orchestrator for that property
attributes	InstanceAttribute	0..N	Corresponds to the attributes defined in the group type and holds the actual attribute values. Each attribute value indicates the value at the time the attribute was accessed. Attributes reflect the state of the underlying group and may change at any time.
capabilities	InstanceCapability	0..N	Corresponds to the capabilities defined in the group type and desired values specified in the group template. Each capability property value indicates the final value used by the orchestrator for that property
requirements	InstanceRequirement	0..N	Maps the source instance in a requirement relationship to the target instances. The mapping is 1 to N and does not have to be 1-1 or onto with respect to all source and target instances.

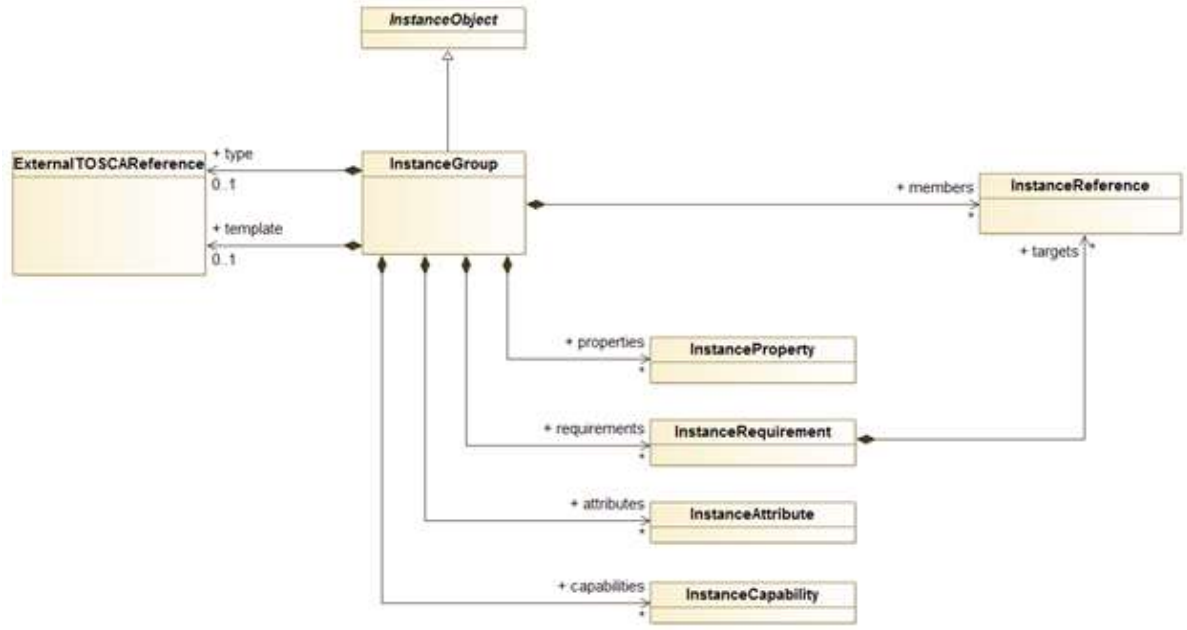


Figure 6 Structure of InstanceGroups, shown as a UML class diagram

5.6.7 InstanceMetadata Definition

InstanceMetadata consists of named objects (a map: name → InstanceObject). Because InstanceObjects can encoded arbitrary information, virtually any information can be added to the instance model as InstanceMetadata.

The instance model version is required (e.g., tosca_instance_version: tosca_simple_yaml_1_0)

Orchestrators may add any additional metadata they like, such as their name and version, licensing, location, signatures, etc.

InstanceMetdata should really be part of InstanceObject so any entity in the instance model can encode any information (like annotations) as InstanceObjects.

5.6.8 InstanceInput Definition

For each input in the service template, and InstanceInput instance is created containing the value used by the orchestrator for the input.

Each InstanceInput is a <name, value> pair consisting of the name of the input in the respective TOSCA input definition and the value of the input encoded for the respective type of the input in the TOSCA input definition.

5.6.9 InstancePolicy Definition

For each policy applied in the service template, an InstancePolicy instance is created and, if applicable, references from the InstancePolicy to the instance model entities the policy is matched are created.

Note that not all policies may be used by the orchestrator if they don't match with entities in the final service topology created by the orchestrator.

Note that InstancePolicies appearance denote that the policy is "applicable" to the service deployment. Their presence in the instance model does not mean they have or will trigger. The objective is to enable the instance model consumer to determine the policies, their configuration, and targeted entities.

Name	Type	Cardinality	Description
template	ExternalToscaReference	0..1 mutually exclusive with type	Allows navigation to the template used to create the instance.
type	ExternalToscaReference	0..1 mutually exclusive with template	Used when the node instance was not created with a template.
targets	InstanceReference	0..N	References the InstanceNodes which the policy applies to
properties	InstanceProperty	0..N	Corresponds to the properties defined in the policy type and desired values specified in the policy template. Each property value indicates the final value used by the orchestrator for that property

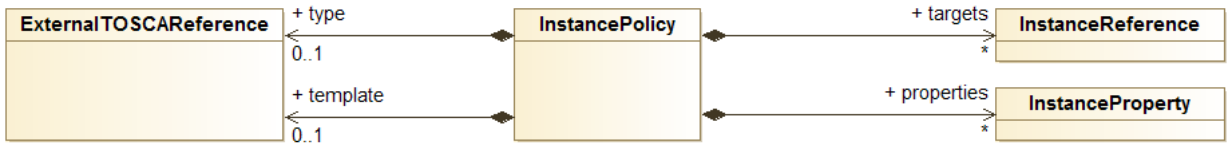


Figure 7 Structure of InstancePolicies displayed as a UML class diagram

5.6.10 InstanceOutput Definition

For each input in the service template, and InstanceOutput instance is created containing the name and value of the Output was created during the orchestration.

Each InstanceOutput is a <name, value> pair consisting of the name of the output in the respective TOSCA output definition and the value of the output encoded for the respective type of the output in the TOSCA input definition.

5.7 Complete Derivation to YAML syntax (textual)

The section describes the complete (top-down) imperative derivation of a deployed service template. It is assumed that orchestration has completed and service template deployment topology is in a steady state (i.e., the in-memory representation of the service template is non-changing) so that the instance model can be derived.

```
Function emit_service_template (serviceTemplate : tosca.types.ServiceTemplate) {
  emit “tosca_instance_version: “, “tosca_simple_yaml_1_0”

  emit “template: “ serviceTemplate.uri

  emit “input_bindings:”
    for each serviceTemplate.topology_template.inputs i
      emit i.name, “: ”, i.value

  emit “node_instances:”
    for each serviceTemplate.rootContainerInstance rootNode
```

```
        emit_nodes(rootNode)
}
```

5.8 Specific derivations

5.8.1 Entities (Graph vertices)

5.8.1.1 Template (from node type)

5.8.1.2 Instance (from template)

5.8.1.3 Instance (from type)

5.8.2 Relationships (graph edges)

5.8.2.1 Relation from between a pair of nodes

5.8.2.2 Mapping M to N (by intension)

5.8.2.3 Explicit M to N (by extension)

5.9 Encoding of information

- Consider 2 approaches
  - DSL-like encoding
    - Instance model has structure similar to service template
  - Object tree encoding
    - Instance model is encoded more naturally as on object tree
- Both are information preserving encodings
  - So it’s possible to translated between them
- There was more interest in the Object tree encoding
  - More natural traversal
  - More natural representation
  - Extensibility in terms of objects (you can represent anything you want) versus DSL

5.10 Serialization Issues

- Users may want to include information defined from other standards/meta models
- Implementations may want to include additional information in TOSCA instances
  - E.g. resource mapping information
  - Structural information indicating relationships among other entities
- Implementations may want to include information not part of TOSCA instances
- TOSCA does not have all the features to support object trees
  - Referencing entities in other subtrees (can only reference specific instances by name, no collections)

- Generalized referencing that work in document, cross document with relative and absolute URIs
- Collections
  - Referencing entities in ordered collections
  - Referencing entities by key

5.11 Relationships between Node Instances

5.12 Recovering the Service Template

Concrete service template

Abstract/portable service template

6 Tracking State Changes

6.1 Reflecting Node Status in the Instance Model

6.2 Visibility of Nodes and Node Attributes

6.3 Orchestration Execution Status

6.4 Update Concurrency

7 Query and Navigation

7.1 Imperative Workflow Use Case

7.2 Imperative interaction with the Instance Model

7.3 State tracking/synchronization

Deterministic tracking/querying of state

7.4 Query

7.5 Navigation

8 Extensibility

8.1 Resource Metadata

Add information to the Instance Model to enabled correlation of TOSCA instances with actual resources.

8.2 Event Triggered Updates

Trigger Instance Model decoration from internal TOSCA runtime and external events.

9 Instance Model Access

9.1 YAML Dump

9.2 Instance Model Access via API

10 Security Considerations

10.1 Access Control

10.2 Authorization

10.3 Obfuscation of Sensitive Information

11 Conformance

The last numbered section in the specification must be the Conformance section. Conformance Statements/Clauses go here.

See “Guidelines to Writing Conformance Clauses”: http://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html.

11.1 Conformance Clause 1: Deployment Orchestration Only

Only needs to comply with the Simple YAML Profile. Includes support of imperative workflows as defined there.

11.2 Conformance Clause 2: Post Orchestration Instance Model Snapshots

The TOSCA Orchestrator dumps a YAML representation, as defined in section 5 Model Structure.

11.3 Conformance Clause 3: Dynamic and Extensible Instance Model Updates

The Instance Model should reflect the changes relating to the deployed topology over time.

11.4 Conformance Clause 4: Programmatic Read-Only Access

Programmatically query and navigate the entire Instance Model.

11.5 Conformance Clause 5: Programmatic Updates

Programatic changes NOT coordinated by the Orchestrator (e.g. no declarative or imperative side-effects using TOSCA DSL)

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Alessandro Rossini (alessandro.rossini@sintef.no), SINTEF

Chris Lauwers (lauwers@ubicity.com), Ubicity

Derek Palma (dpalma@vnomic.com), Vnomic

Franck Chauvel (Franck.Chauvel@sintef.no), SINTEF

Matt Rutkowski (mrutkows@us.ibm.com), IBM

Lawrence Lamers (ljlamers@vmware.com), VMware

Appendix B. Schema Modeling Language

We describe in the following sections the schema of the TOSCA instance model using a minimal object-oriented notation, inspired by eMOF [ref] and one of its implementation called ECore. This is our M3-language (i.e., our meta-metamodel), which we illustrate on **Error! Reference source not found.**, below. Although we do reuse the UML class diagram notation to illustrate specific part of this schema, please note that our meta-metamodel is not the one behind the UML notation.

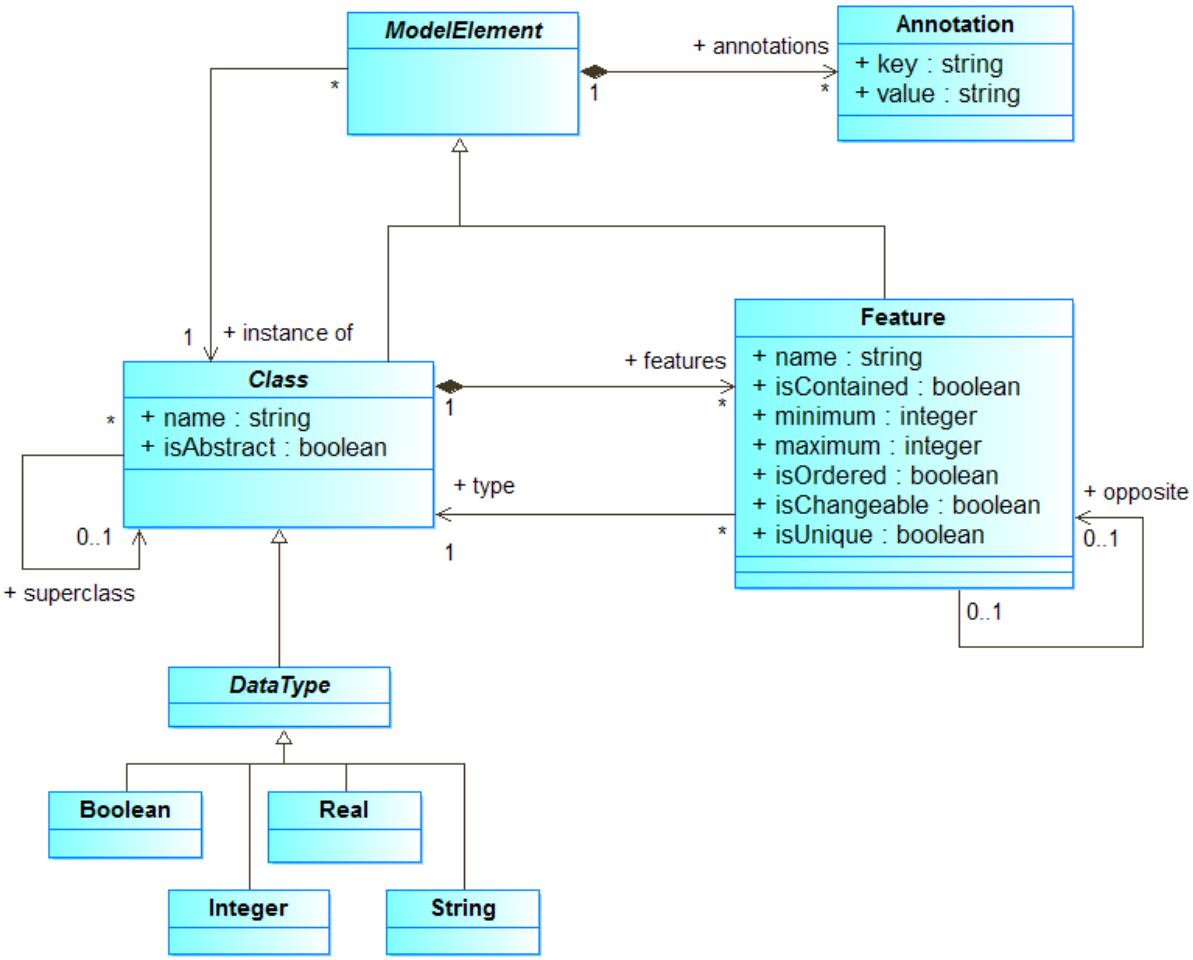


Figure 8 Concepts used to describe the TOSCA instance model schema, depicted as a UML class diagram.

For instance, visibility levels, which UML denotes as '+', '-' or '#', do not exist in our M3-language and should not be taken into account, consequently. Consider for instance **Error! Reference source not found.**, which shows an excerpt of the structure of an InstanceNode. Here an InstanceNode may originate from a template (another InstanceNode) and contains several InstanceCapability and several InstanceRequirement. Note the use of colors: Blue identifies the concept of the M3-language, whereas yellow identifies the instance model concepts.

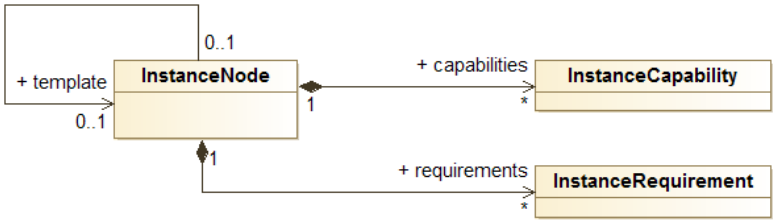


Figure 9 Excerpt of the TOSCA instance model schema, depicted as a UML class diagram

Although this is an UML class diagram, we understand it as an instantiation of our M3-language, (see **Error! Reference source not found.**), that is the set of objects shown by **Error! Reference source not found.**, below. It shows an UML object diagram that contains all the instances of our M3-concepts that we need to describe the schema fragment showed by **Error! Reference source not found.**.

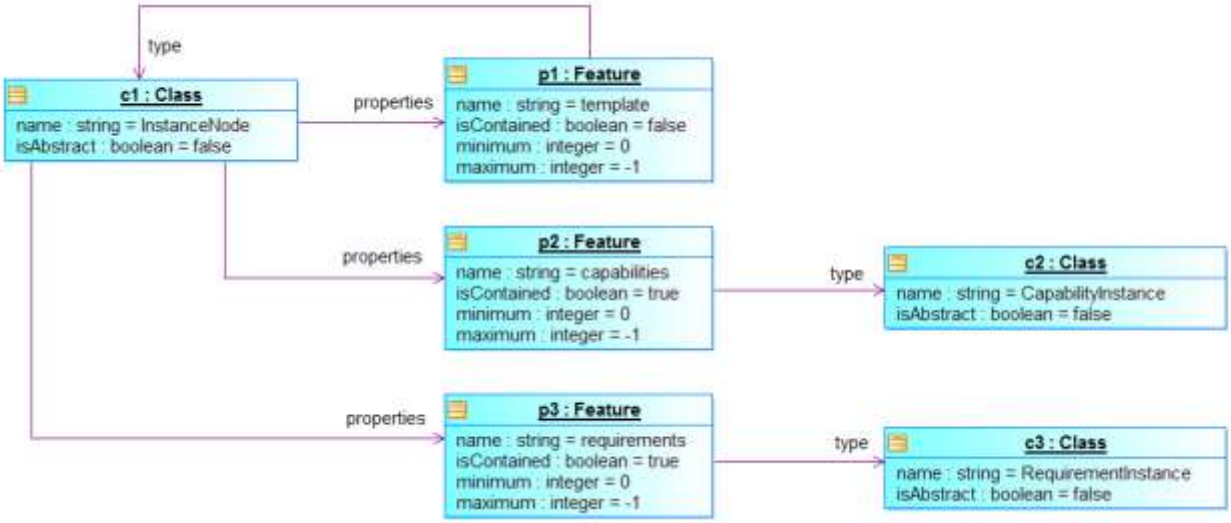


*Figure 10 Instances of the M3-language (see **Error! Reference source not found.**) that reify the concepts shown in **Error! Reference source not found.**, depicted as a UML object diagram*

11.6 Collections

Ordering is a consumption issue and is required to support determinism and comparison

Ordered -> Lists

Unordered -> Bags and Sets

11.7 TOSCA DSL Entity References

The instance model encoding allows instance model entities to reference entities in TOSCAL DSL artifacts in order to avoid translating all TOSCA DSL entities into instance model entities.

11.7.1 ToscaDslRef Definition

References an entity encoded in the TOSCA DSL. The reference indicates the specific TOSCA artifact (e.g. YAML document) and the identity or path of the target concept (node type, node template, etc.), and slot (property, attribute etc.)

11.8 Cross Instance Model Linking

11.8.1 URIs and Entity Identity

Each entity has a unique URI, scoped (going outward):

Deployment ID (service template instance name)

Tenant/Customer ID

TOSCA runtime instance ID

11.8.2 Local Model Linking

Linking instance models in same TOSCA runtime instances.

11.8.3 Foreign Model Linking

Linking instance models across different TOSCA runtimes instances.

Appendix C. Revision History

Revision	Date	Editor	Changes Made
WD01, Rev06	10/2017	Derek Palma, Vnomic	Initial version for TC review