

# Topology and Orchestration Specification for Cloud Applications Version 1.0

## Committee Specification Draft 01

08 March 2012

### Specification URIs

#### This version:

<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd01/TOSCA-v1.0-csd01.doc> (Authoritative)  
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd01/TOSCA-v1.0-csd01.html>  
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd01/TOSCA-v1.0-csd01.pdf>

#### Previous version:

N/A

#### Latest version:

<http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.doc> (Authoritative)  
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html>  
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf>

### Technical Committee:

[OASIS Topology and Orchestration Specification for Cloud Applications \(TOSCA\) TC](#)

#### Chairs:

Paul Lipton ([paul.lipton@ca.com](mailto:paul.lipton@ca.com)), CA Technologies  
Simon Moser ([smoser@de.ibm.com](mailto:smoser@de.ibm.com)), IBM

#### Editors:

Arvind Srinivasan ([arvindsr@us.ibm.com](mailto:arvindsr@us.ibm.com)), IBM  
Thomas Spatzier ([thomas.spatzier@de.ibm.com](mailto:thomas.spatzier@de.ibm.com)), IBM

### Declared XML namespace:

- <http://docs.oasis-open.org/tosca/ns/2011/12>

### Abstract:

This specification introduces the formal description of Service Templates, including their structure, properties, and behavior.

The concept of a “service template” is used to specify the “topology” (or structure) and “orchestration” (or invocation of management behavior) of IT services. Typically, services are provisioned in an IT infrastructure and their management behavior must be orchestrated in accordance with constraints or policies, for example to achieve service level objectives.

### 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.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the

“[Send A Comment](http://www.oasis-open.org/committees/tosca/)” button on the Technical Committee’s web page at <http://www.oasis-open.org/committees/tosca/>.

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 Technical Committee web page (<http://www.oasis-open.org/committees/tosca/ipr.php>).

**Citation format:**

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

**[TOSCA-v1.0]**

*Topology and Orchestration Specification for Cloud Applications Version 1.0*. 08 March 2012.  
OASIS Committee Specification Draft 01. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd01/TOSCA-v1.0-csd01.html>.

---

# Notices

Copyright © OASIS Open 2012. All Rights Reserved.

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 <http://www.oasis-open.org/who/trademark.php> for above guidance.

---

# Table of Contents

1	Introduction .....	6
2	Language Design .....	7
2.1	Dependencies on Other Specifications .....	7
2.2	Notational Conventions .....	7
2.3	Normative References .....	7
2.4	Non-Normative References .....	8
2.5	Namespaces .....	8
2.6	Language Extensibility .....	8
2.7	Overall Language Structure .....	8
2.7.1	Syntax .....	9
2.7.2	Properties .....	9
3	Core Concepts and Usage Pattern .....	13
3.1	Core Concepts .....	13
3.2	Use Cases .....	14
3.2.1	Services as Marketable Entities .....	14
3.2.2	Portability of Service Templates .....	15
3.2.3	Service Composition .....	15
3.2.4	Relation to Virtual Images .....	15
4	Node Types .....	16
4.1	Syntax .....	16
4.2	Properties .....	17
4.3	Derivation Rules .....	19
4.4	Example .....	20
5	Relationship Types .....	22
5.1	Syntax .....	22
5.2	Properties .....	22
5.3	Example .....	23
6	Topology Template .....	24
6.1	Syntax .....	24
6.2	Properties .....	26
6.3	Example .....	29
7	Plans .....	31
7.1	Syntax .....	31
7.2	Properties .....	31
7.3	Use of Process Modeling Languages .....	32
7.4	Example .....	32
8	Security Considerations .....	34
9	Conformance .....	35
Appendix A.	Portability and Interoperability Considerations .....	36
Appendix B.	Complete TOSCA Grammar .....	37
Appendix C.	TOSCA Schema .....	42
Appendix D.	Sample .....	53
D.1	Sample Service Topology Definition .....	53

Appendix E.	Revision History .....	57
-------------	------------------------	----

---

# 1 Introduction

IT services (or just *services* in what follows) are the main asset within IT environments in general, and in cloud environments in particular. The advent of cloud computing suggests the utility of standards that enable the (semi-) automatic creation and management of services (a.k.a. service automation). These standards describe a service and how to manage it independent of the supplier creating the service and independent of any particular cloud provider and the technology hosting the service. Making service topologies (i.e. the individual components of a service and their relations) and their orchestration plans (i.e. the management procedures to create and modify a service) interoperable artifacts, enables their exchange between different environments. This specification explains how to define services in a portable and interoperable manner in a *Service Template* document.

---

## 2 Language Design

The TOSCA language introduces a grammar for describing service templates by means of Topology Templates and plans. The focus is on design time aspects, i.e. the description of services to ensure their exchange. Runtime aspects are addressed by providing a container for specifying models of plans which support the management of instances of services.

The language provides an extension mechanism that can be used to extend the definitions with additional vendor-specific or domain-specific information.

### 2.1 Dependencies on Other Specifications

TOSCA utilizes the following specifications:

- WSDL 1.1
- XML Schema 1.0

and relates to:

- OVF 1.1

### 2.2 Notational Conventions

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].

### 2.3 Normative References

- |                     |   |
|---------------------|---|
| [RFC2119]           | S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> , IETF RFC 2119, March 1997.              |
| [BPEL 2.0]          | OASIS Web Services Business Process Execution Language (WS-BPEL) 2.0, <a href="http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf">http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf</a>               |
| [BPMN 2.0]          | OMG Business Process Model and Notation (BPMN) Version 2.0 - Beta 1, <a href="http://www.omg.org/spec/BPMN/2.0/">http://www.omg.org/spec/BPMN/2.0/</a>  |
| [OVF]               | Open Virtualization Format Specification Version 1.1.0, <a href="http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf">http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf</a> |
| [WSDL 1.1]          | Web Services Description Language (WSDL) Version 1.1, W3C Note, <a href="http://www.w3.org/TR/2001/NOTE-wsdl-20010315">http://www.w3.org/TR/2001/NOTE-wsdl-20010315</a>                                       |
| [XML Infoset]       | XML Information Set, W3C Recommendation, <a href="http://www.w3.org/TR/2001/REC-xml-infoset-20011024/">http://www.w3.org/TR/2001/REC-xml-infoset-20011024/</a>  |
| [XML Namespaces]    | Namespaces in XML 1.0 (Second Edition), W3C Recommendation, <a href="http://www.w3.org/TR/REC-xml-names/">http://www.w3.org/TR/REC-xml-names/</a>   |
| [XML Schema Part 1] | XML Schema Part 1: Structures, W3C Recommendation, October 2004, <a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a>  |
| [XML Schema Part 2] | XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, <a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a>   |
| [XMLSpec]           | XML Specification, W3C Recommendation, February 1998, <a href="http://www.w3.org/TR/1998/REC-xml-19980210">http://www.w3.org/TR/1998/REC-xml-19980210</a>   |
| [XPath 1.0]         | XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, <a href="http://www.w3.org/TR/1999/REC-xpath-19991116">http://www.w3.org/TR/1999/REC-xpath-19991116</a>                             |

## 2.4 Non-Normative References

## 2.5 Namespaces

This specification uses a number of namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [XML Namespaces]). Furthermore, the namespace <http://docs.oasis-open.org/tosca/ns/2011/12> is assumed to be the default namespace, i.e. the corresponding namespace name `ste` is omitted in this specification to improve readability.

Prefix	Namespace
ste	<a href="http://docs.oasis-open.org/tosca/ns/2011/12">http://docs.oasis-open.org/tosca/ns/2011/12</a>
xs	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>
wsdl	<a href="http://schemas.xmlsoap.org/wsdl/">http://schemas.xmlsoap.org/wsdl/</a>
bpmn	<a href="http://www.omg.org/bpmn/2.0">http://www.omg.org/bpmn/2.0</a>

**Table 1** Prefixes and namespaces used in this specification

All information items defined by TOSCA are identified by one of the XML namespace URIs above [XML Namespaces]. A normative XML Schema [XML Schema Part 1, XML Schema Part 2] document for TOSCA can be obtained by dereferencing one of the XML namespace URIs.

## 2.6 Language Extensibility

The TOSCA extensibility mechanism allows:

- Attributes from other namespaces to appear on any TOSCA element
- Elements from other namespaces to appear within TOSCA elements
- Extension attributes and extension elements **MUST NOT** contradict the semantics of any attribute or element from the TOSCA namespace

The specification differentiates between mandatory and optional extensions (the section below explains the syntax used to declare extensions). If a mandatory extension is used, a compliant implementation **MUST** understand the extension. If an optional extension is used, a compliant implementation **MAY** ignore the extension.

## 2.7 Overall Language Structure

A *Service Template* is an XML document that consists of a Topology Template, Node Types, Relationship Types and Plans. This section explains the overall structure of a Service Template, the extension mechanism, and import features. Later sections describe in detail Topology Templates, Node Types, Relationship Types and Plans.



## 2.7.1 Syntax

```
1 <ServiceTemplate id="ID"
2     name="string"?
3     targetNamespace="anyURI">
4
5     <Extensions>?
6         <Extension namespace="anyURI"
7             mustUnderstand="yes|no"?/>+
8     </Extensions>
9
10    <Import namespace="anyURI"?
11        location="anyURI"?
12        importType="anyURI"/>*
13
14    <Types>?
15        <xs:schema .../>*
16    </Types>
17
18    (
19        <TopologyTemplate>
20            ...
21        </TopologyTemplate>
22    |
23        <TopologyTemplateReference reference="xs:QName">
24    ) ?
25
26    <NodeTypes>?
27        ...
28    </NodeTypes>
29
30    <RelationshipTypes>?
31        ...
32    </RelationshipTypes>
33
34    <Plans>?
35        ...
36    </Plans>
37
38 </ServiceTemplate>
```

## 2.7.2 Properties

The `ServiceTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Service Template. The identifier of the Service Template **MUST** be unique within the target namespace.

**Note:** For elements defined in this specification, the value of the `id` attribute of an element is used as the local name part of the fully-qualified name (QName) of that element, by which it can be referenced from within another definition.

- `name`: This optional attribute specifies the name of the Service Template.

**Note:** The `name` attribute for elements defined in this specification can generally be used as descriptive, human-readable name.

- `targetNamespace`: The value of this attribute is the namespace for the Service Template.
- `Extensions`: This element specifies namespaces of TOSCA extension attributes and extension elements. The element is optional.

If present, the `Extensions` element MUST include at least one `Extension` element. The `Extension` element is used to specify a namespace of TOSCA extension attributes and extension elements, and indicates whether they are mandatory or optional.

The attribute `mustUnderstand` is used to specify whether the extension must be understood by a compliant implementation. If the `mustUnderstand` attribute has value “yes” (which is the default value for this attribute) the extension is mandatory. Otherwise, the extension is optional. If a TOSCA implementation does not support one or more of the extensions with `mustUnderstand="yes"`, then the Service Template MUST be rejected. Optional extensions MAY be ignored. It is not necessary to declare optional extensions.

The same extension URI MAY be declared multiple times in the `Extensions` element. If an extension URI is identified as mandatory in one `Extension` element and optional in another, then the mandatory semantics have precedence and MUST be enforced. The extension declarations in an `Extensions` element MUST be treated as an unordered set.
- `Import`: This element declares a dependency on external Service Template, XML Schema definitions, or WSDL definitions. Any number of `Import` elements MAY appear as children of the `ServiceTemplate` element.

The `namespace` attribute specifies an absolute URI that identifies the imported definitions. This attribute is optional. An `Import` element without a `namespace` attribute indicates that external definitions are in use, which are not namespace-qualified. If a `namespace` attribute is specified then the imported definitions MUST be in that namespace. If no namespace is specified then the imported definitions MUST NOT contain a `targetNamespace` specification. The namespace `http://www.w3.org/2001/XMLSchema` is imported implicitly. Note, however, that there is no implicit XML Namespace prefix defined for `http://www.w3.org/2001/XMLSchema`.

The `location` attribute contains a URI indicating the location of a document that contains relevant definitions. The location URI MAY be a relative URI, following the usual rules for resolution of the URI base [XML Base, RFC 2396]. The `location` attribute is optional. An `Import` element without a `location` attribute indicates that external definitions are used but makes no statement about where those definitions might be found. The `location` attribute is a hint and a TOSCA compliant implementation is not obliged to retrieve the document being imported from the specified location.

The mandatory `importType` attribute identifies the type of document being imported by providing an absolute URI that identifies the encoding language used in the document. The value of the `importType` attribute MUST be set to `http://docs.oasis-open.org/tosca/ns/2011/12` when importing Service Template documents, to `http://schemas.xmlsoap.org/wsdl/` when importing WSDL 1.1 documents, and to `http://www.w3.org/2001/XMLSchema` when importing an XSD document.

According to these rules, it is permissible to have an `Import` element without `namespace` and `location` attributes, and only containing an `importType` attribute. Such an `Import` element indicates that external definitions of the indicated type are in use that are not namespace-qualified, and makes no statement about where those definitions might be found.

A Service Template MUST define or import all Topology Template, Node Types, Relationship Types, Plans, WSDL definitions, and XML Schema documents it uses. In order to support the use of definitions from namespaces spanning multiple documents, a Service Template MAY include more than one import declaration for the same namespace and `importType`. Where a service template has more than one import declaration for a given namespace and `importType`, each declaration MUST include a different location value. `Import` elements are conceptually unordered. A Service Template MUST be rejected if the imported documents contain conflicting definitions of a component used by the importing Service Template.

Documents (or namespaces) imported by an imported document (or namespace) are not transitively imported by a TOSCA compliant implementation. In particular, this means that if an external item is used by an element enclosed in the Service Template, then a document (or namespace) that defines that item **MUST** be directly imported by the Service Template. This requirement does not limit the ability of the imported document itself to import other documents or namespaces.

- **Types**: This element specifies XML definitions introduced within the Service Template document. Such definitions are provided within one or more separate Schema Definitions (usually `xs:schema` elements). The **Types** element defines XML definitions within a Service Template file without having to define these XML definitions in separate files and import them. Note, that an `xs:schema` element nested in the **Types** element **MUST** be a valid XML schema definition. In case the `targetNamespace` attribute of a nested `xs:schema` element is not specified, all definitions within this element become part of the target namespace of the encompassing **ServiceTemplate** element.

**Note**: The specification supports the use of any type system nested in the **Types** element. Nevertheless, only the support of `xs:schema` is **REQUIRED** from any compliant implementation.

- **TopologyTemplate**: This element specifies in place the topological structure of an IT service by means of a directed graph.

The main ingredients of a Topology Template are a set of Node Templates and Relationship Templates. The Node Templates are the nodes of the directed graph. The Relationship Templates are the directed edges between the nodes; each indicates the semantics of the corresponding relationships.

- **TopologyTemplateReference**: This element references a Topology Template. Its `reference` attribute specifies the QName of the definition available by reference in the document under definition. The namespace of the referenced Topology Template **MUST** be imported into the Service Template by means of an **Import** element.

**Note** that either zero or one Topology Template **MUST** occur in a Service Template, either defined in place via a **TopologyTemplate** element or referenced via a **TopologyTemplateReference**.

- **NodeTypes**: This element specifies the types of Node (Templates), i.e., their properties and behavior.
- **RelationshipTypes**: This element specifies the types of relationships, i.e. the kind of links between Node Templates within a Service Template, and their properties.
- **Plans**: This element specifies the operational behavior of the service. Each **Plan** contained in the **Plans** element specifies how to create, terminate or manage the service.

A Service Template document can be intended to be instantiated into a service instance or it can be intended to be composed into other Service Templates. A Service Template document intended to be instantiated **MUST** contain either a **TopologyTemplate** or a **TopologyTemplateReference**, but not both. A Service Template document intended to be composed **MUST** include at least one of a **NodeTypes**, **RelationshipTypes**, or **Plans** element. This technique supports a modular definition of Service Templates. For example, one document can contain only Node Types that are referenced by a Service Template document that contains just a Topology Template and Plans. Similarly, Node Type Properties can be defined in separate XML Schema Definitions that are imported and referenced when defining a Node Type.

Example of the use of a type definition:

```
<Types>
```

```

231 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
232         elementFormDefault="qualified"
233         attributeFormDefault="unqualified">
234   <xs:element name="ProjectProperties">
235     <xs:complexType>
236       <xs:sequence>
237         <xs:element name="Owner" type="xs:string"/>
238         <xs:element name="ProjectName" type="xs:string"/>
239         <xs:element name="AccountID" type="xs:string"/>
240       </xs:sequence>
241     </xs:complexType>
242   </xs:element>
243 </xs:schema>
244 </Types>

```

245 All TOSCA elements MAY use the `documentation` element to provide annotation for users. The  
 246 content could be a plain text, HTML, and so on. The `documentation` element is optional and has the  
 247 following syntax:

```

248 1 <documentation source="anyURI"? xml:lang="language"?>
249 2   ...
250 3 </documentation>

```

251 Example of use of a documentation:

```

252 <ServiceTemplate id="myService" name="My Service" ...>
253
254   <documentation xml:lang="EN">
255     This is a simple example of the usage of the documentation
256     element as nested under a ServiceTemplate element.
257   </documentation>
258
259 </ServiceTemplate>

```

## 3 Core Concepts and Usage Pattern

The main concepts behind TOSCA are described and some usage patterns of Service Templates are sketched.

### 3.1 Core Concepts

This specification defines a *metamodel* for defining IT services. This metamodel defines both the structure of a service as well as how to manage it. A *Topology Template* (also referred to as the *topology model* of a service) defines the *structure* of a service. *Plans* define the process models that are used to create and terminate a service as well as to manage a service during its whole lifetime. The major artifacts defining a service are depicted in Figure 1.

A Topology Template consists of a set of Node Templates and Relationship Templates that together define the topology model of a service as a (not necessarily connected) directed graph. A node in this graph is represented by a *Node Template*. A Node Template specifies the occurrence of a Node Type as a component of a service. A *Node Type* defines the properties of such a component (via *Node Type Properties*) and the operations (via *Interfaces*) available to manipulate the component. Node Types are defined separately for reuse purposes and a Node Template references a Node Type and adds usage constraints, such as how many times the component can occur.

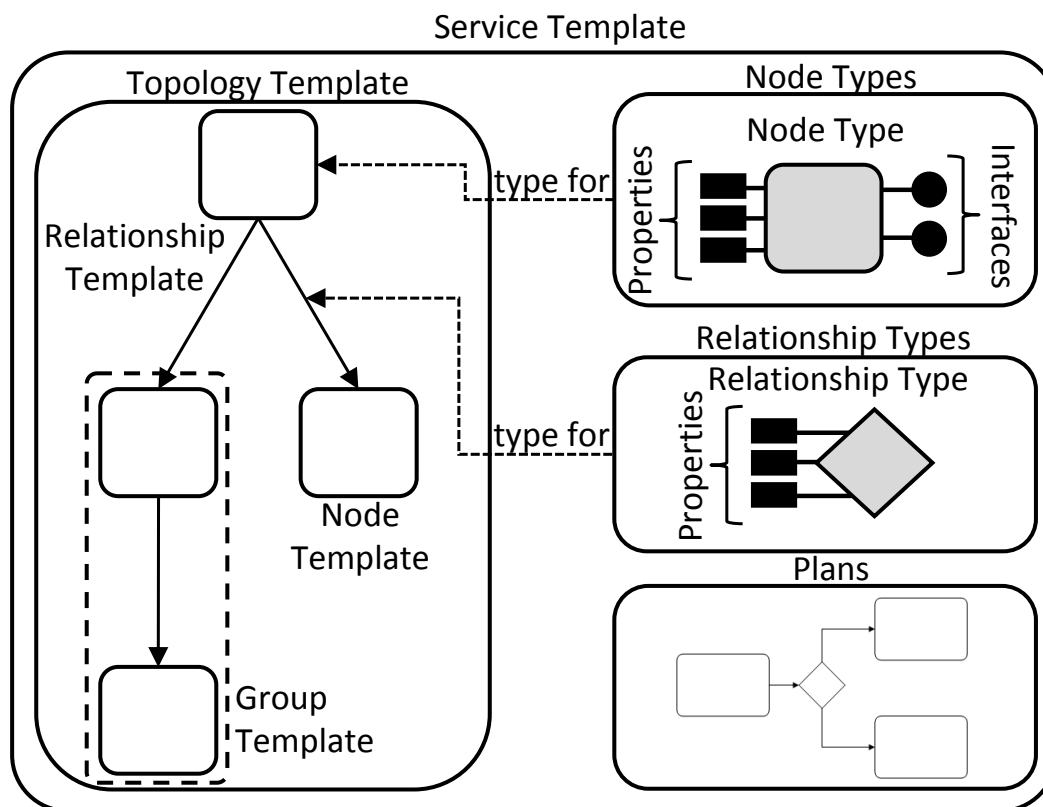


Figure 1: Structural Elements of a Service Template and their Relations

For example, consider a service that consists of an application server, a process engine, and a process model. A Topology Template defining that service would include one Node Template of Node Type “application server”, another Node Template of Node Type “process engine”, and a third Node Template of Node Type “process model”. The application server Node Type defines properties like the IP address

of an instance of this type, an operation for installing the application server with the corresponding IP address, and an operation for shutting down an instance of this application server. A constraint in the Node Template can specify a range of IP addresses available when making a concrete application server available.

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 and any properties of the relationship. Relationship Types are defined separately for reuse purposes. The Relationship Template indicates the types of nodes and the direction of the relationship by defining one source and one target Node Template (in nested *SourceNodeTemplate* and *TargetNodeTemplate* elements). The Relationship Template also defines any constraints with the optional *RelationshipConstraints* element.

For example, a relationship can be established between the process engine Node Template and application server Node Template with the meaning “hosted by”, and between the process model Node Template and process engine Node Template with meaning “deployed on”.

A deployed service is an 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. The build plan will provide actual values for the various properties of the various Node Templates and Relationship Templates of the Topology Template. These values can come from input passed in by users as triggered by human interactions defined within the build plan, by automated operations defined within the build plan (such as a directory lookup), or the templates can specify default values for some properties. The build plan will typically make use of operations of the Node Types of the Node Templates.

For example, the application server Node Template will be instantiated by installing an actual application server at a concrete IP address considering the specified range of IP addresses. Next, the process engine Node Template will be instantiated by installing a concrete process engine on that application server (as indicated by the “hosted by” relationship template). Finally, the process model Node Template will be instantiated by deploying the process model on that process engine (as indicated by the “deployed on” relationship template).

*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 any other interface (e.g. 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.

## 3.2 Use Cases

The specification supports at least the following major use cases.

### 3.2.1 Services as Marketable Entities

Standardizing Service Templates will support the creation of a market for hosted IT services. Especially, a standard for specifying Topology Templates (i.e. the set of components a service consists of as well as their mutual dependencies) enables interoperable definitions of the structure of services. Such a service topology model could be created by a service developer who understands the internals of a particular service. The Service Template could then be published in catalogs of one or more service providers for selection and use by potential customers. Each service provider would map the specified service topology to its available concrete infrastructure in order to support concrete instances of the service and adapt the management plans accordingly.

Making a concrete instance of a Topology Template can be done by running a corresponding Plan (so-called instantiating management plan, a.k.a. 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.

Thus, not only the structure of a service can be defined in an interoperable manner, but also its management plans. These Plans describe how instances of the specified service are created and managed. Defining a set of management plans for a service will significantly reduce the cost of hosting a service by providing reusable knowledge about best practices for managing each service. While the modeler of a service can include deep domain knowledge into a plan, the user of such a service can use a plan by simply “invoking” it. This hides the complexity of the underlying service behavior. This is very similar to the situation resulting in the specification of ITIL.

### 3.2.2 Portability of Service Templates

Standardizing Service Templates supports the portability of definitions of IT Services. Here, portability denotes the ability of one cloud provider to understand the structure and behavior of a Service Template created by another party, e.g. another cloud provider, enterprise IT department, or service developer.

Note that portability of a service does not imply portability of its encompassed components. Portability of a service means that its definition can be understood in an interoperable manner, i.e. the topology model and corresponding plans are understood by standard compliant vendors. Portability of the individual components themselves making up a particular service has to be ensured by other means – if it is important for the service.

### 3.2.3 Service Composition

Standardizing Service Templates facilitates composing a service from components even if those components are hosted by different providers, including the local IT department, or in different automation environments, often built with technology from different suppliers. For example, large organizations could use automation products from different suppliers for different data centers, e.g., because of geographic distribution of data centers or organizational independence of each location. A Service Template provides an abstraction that does not make assumptions about the hosting environments.

### 3.2.4 Relation to Virtual Images

A cloud provider can host a service based on virtualized middleware stacks. These middleware stacks might be represented by an image definition such as an OVF [OVF] package. If OVF is used, a node in a Service Template can correspond to a virtual system or a component (OVF's "product") running in a virtual system, as defined in an OVF package. If the OVF package defines a virtual system collection containing multiple virtual systems, a sub-tree of a Service Template could correspond to the OVF virtual system collection.

A Service Template provides a way to declare the association of Service Template elements to OVF package elements. Such an association expresses that the corresponding Service Template element can be instantiated by deploying the corresponding OVF package element. These associations are not limited to OVF packages. The associations could be to other package types or to external service interfaces. This flexibility allows a Service Template to be composed from various virtualization technologies, service interfaces, and proprietary technology.



## 4 Node Types

This chapter specifies how *Node Types* are defined. A Node Type is a reusable entity that defines the type of one or more Node Templates. As such, a Node Type defines observable properties via *Node Type Properties*. A Node Type can inherit properties from another Node Type by means of the *DerivedFrom* element. The functions that can be performed on (an instance of) a corresponding Node Template are defined by the *Interfaces* of the Node Type. Finally, interfaces supporting management Policies are defined for a Node Type.

### 4.1 Syntax

```
<NodeTypes>?
  <NodeType id="ID"
    name="string"?>+
    <NodeTypeProperties element="QName"?
      type="QName"?/>?
    <DerivedFrom nodeTypeRef="QName"/>?
    <InstanceStates>?
      <InstanceState state="anyURI">+
    </InstanceStates>
    <Interfaces>?
      <Interface>+
        (
          <WSDL portType="QName"
            operation="NCName"?>+
          |
          <REST method="GET | PUT | POST | DELETE"
            requestURI="anyURI"
            requestPayload="QName"?
            responsePayload="QName"?>+
          |
          <Operation name="NCame">+
        )
        <InputParameters>?
          <InputParamter name="string"
            type="string"
            required="yes|no">+
        </InputParameters>
        <OutputParameters>?
          <OutputParamter name="string"
            type="string"
            required="yes|no">+
        </OutputParameters>
```



```

426 45
427 46         <Implementations>
428 47
429 48         <Implementation implementationID="anyURI"?
430 49             language="anyURI"?>+
431 50             (
432 51                 <ImplementationProper>?
433 52                     code
434 53                 </ImplementationProper>
435 54                 |
436 55                 <ImplementationReference ref="anyURI"/>?
437 56             )
438 57         <Implementation>
439 58
440 59         </Implementations>
441 60     </Operation>
442 61 )
443 62
444 63 </Interface>
445 64
446 65 </Interfaces>
447 66
448 67 <Policies>?
449 68     <Policy name="string" type="anyURI">+
450 69         policy specific content
451 70     </Policy>
452 71 </Policies>
453 72
454 73 <DeploymentArtifacts>?
455 74     <DeploymentArtifact name="string" type="anyURI">+
456 75         artifact specific content
457 76     </DeploymentArtifact>
458 77 </DeploymentArtifacts>
459 78
460 79 </NodeType>
461 80
462 81 </NodeTypes>

```

## 463 4.2 Properties

464 The `NodeType` element has the following properties:

- 465 • `id`: This attribute specifies the identifier of the Node Type. The identifier of the Node Type MUST  
466 be unique within the target namespace.
- 467 • `name`: This optional attribute specifies the name of the Node Type.
- 468 • `NodeTypeProperties`: These are the observable properties of the Node Type, such as its  
469 configuration and state.
- 470 • `DerivedFrom`: This is an optional reference to another Node Type from which this Node Type  
471 derives. Conflicting definitions are resolved by the rule that local new definitions always override  
472 derived definitions. See section 4.3 Derivation Rules for details.
- 473 • `InstanceStates`: This optional element lists the set of states an instance of this Node Type can  
474 occupy at runtime.
- 475 • `Interfaces`: These are the definitions of functions that can be performed on (instances of) this  
476 Node Type.

477       • **Policies:** The nested list of elements provides information related to a particular management  
478       aspect like billing or monitoring.

479       • **DeploymentArtifacts:** This element specifies deployment artifacts relevant for the Node  
480       Type. A deployment artifact is an entity that – if specified – is needed for creating an instance of  
481       the corresponding Node Type. For example, a virtual image could be a deployment artifact of a  
482       JEE server.

483 The **NodeTypeProperties** element has one but not both of the following properties:

484       • The **element** attribute provides the QName of an XML element defining the structure of the  
485       Node Type Properties.

486       • The **type** attribute provides the QName of an XML (complex) type defining the structure of the  
487       Node Type Properties.

488 The **DerivedFrom** element has the following properties:

489       • **nodeTypeRef:** The QName specifies the Node Type from which this Node Type derives its  
490       definitions.

491 The **InstanceStates** element has the following properties:

492       • **InstanceState:** specifies a potential state.

493 The **InstanceState** element has the following properties:

494       • **state:** a URI that represents a potential state.

495 The **Interface** element has one of the following properties:

496       • **WSDL:** specifies a WSDL port type or an operation of a port type as part of the Interface.

497       • **REST:** specifies an HTTP request as part of the Interface.

498       • **Operation:** specifies a proprietary implementation as part of the Interface.

499 The **WSDL** element has the following properties:

500       • **portType:** This is the QName of the port type that contains the definition of one or more  
501       operations defined as part of the Interface. Note that the namespace of the portType **MUST** be  
502       imported.

503       • **operation:** This optional attribute specifies the name of a single operation of the port type to  
504       become part of the Node Type Interface. If this attribute is not specified, the complete WSDL port  
505       type becomes part of the Node Type Interface.

506 The **REST** element has the following properties:

507       • **method:** This is the name of an HTTP method (often in a REST-style Interface).

508       • **requestURI:** This is the requestURI necessary to create the request.

509       • **requestPayload:** The QName specifies the schema of the HTTP message payload passed  
510       with the request to the HTTP processor.

511       • **responsePayload:** The QName specifies the schema of the payload passed with the response  
512       message from the HTTP processor.

513       **Note:** The combination of **method** and **requestURI** **SHOULD** uniquely identify a **REST** element  
514       within the Service Template.

515 The **Operation** element has the following properties:

516       • **name:** This is the name of the operation. The name of an operation **SHOULD** be unique within a  
517       Node Type.

- **InputParameters:** This optional property contains one or more nested `InputParameter` elements. Each such element specifies three attributes: the `name` of the parameter, its `type`, and whether it has to be available as input (`required` attribute with a value of “yes”, which is the default) or not (value “no”).  
Note that the types of the parameters specified for an operation MUST comply with the type systems of the languages of implementations’ proper.
- **OutputParameters:** This optional property contains one or more nested `OutputParameter` elements. Each such element specifies the `name` of the parameter and its `type`.
- **Implementations:** This element contains one or more `Implementation` elements, each of which encompasses either the actual `code` of the implementation of the operation or a reference to the code. The `implementationID` attribute of the `Implementation` element allows for providing different implementations for the same operation – this is necessary because the implementation often depends on the environment the operation will run in. The `language` attribute allows to specify the language of the implementation, e.g. one implementation might be provided as a perl script, another one as php, and so on.

The `Policy` element has the following properties:

- The `type` attribute specifies the kind of policy (e.g. management practice) supported by an instance of the Node Type containing this element. The `name` attribute defines the name of the policy. The name value MUST be unique within a given Node Type containing the current definition of the `Policy`.

Consider a hypothetical billing policy. In this example the type `www.sample.com/BillingPractice` could define a policy for billing usage of a service instance. The policy specific content can define the interface providing the operations to perform billing. Further content could specify the granularity of the base for payment, e.g. it could provide an enumeration with the possible values “service”, “resource”, and “labor”. A value of “service” might specify that an instance of the corresponding node will be billed during its instance lifetime. A value of “resource” might specify that the resources consumed by an instance will be billed. A value of “labor” might specify that the use of a plan affecting a node instance will be billed.

The `DeploymentArtifact` element has the following properties:

- `name`: The attribute specifies the name of the artifact. Note, that uniqueness of the name within the scope of the encompassing Node Type SHOULD be guaranteed by the definition.
- `type`: The attribute specifies the type of the deployment artifact definition that is related to the Node Type, i.e. the attribute gives a hint how to interpret the body of the `DeploymentArtifact` element.  
Note, that the combination of name and type SHOULD be unique within the scope of the Node Type.
- The body of this element contains the type-specific content.

For example, if the `type` attribute contains the value `http://docs.oasis-open.org/tosca/ns/2011/12/deploymentArtifacts/ovfRef`, the body will contain an XML fragment with a reference to an OVF package and a mapping between service template data and elements of the respective OVF envelope.

## 4.3 Derivation Rules

The following rules on combining definitions based on `DerivedFrom` apply:

- Node Type Properties: It is assumed that the XML element (or type) representing the Node Type Properties extends the XML element (or type) of the Node Type Properties of the Node Type referenced in the `DerivedFrom` element.
- Instance States: The set of instance states of this Node Type consists of the set union of the instances states defined by the Nodes Type derived from and the instance states defined by this Node Type. A set of instance states of the same name will be combined into a single instance state of the same name.
- WSDL: The set of WSDL operations of this Node Type consists of the set union of the WSDL operations defined by the Node Type derived from and the WSDL operations defined by the Node Type. A WSDL operation defined by this Node Type substitutes a WSDL operation with the same name and signature of the Node Type derived from.
- REST: The set of REST requests of this Node Type consists of the set union of the REST requests defined by the Nodes Type derived from and the REST requests defined by this Node Type. A REST request defined by this Node Type substitutes a REST request with the same identity of the Node Type derived from.
- Operation: The set of operations of this Node Type consists of the set union of the operations defined by the Nodes Type derived from and the operations defined by this Node Type. An operation defined by this Node Type substitutes an operation with the same name of the Node Type derived from.
- Deployment Artifacts: The set of deployment artifacts of this Node Type consists of the set union of the deployment artifacts defined by the Nodes Type derived from and the deployment artifacts defined by this Node Type. A deployment artifact defined by this Node Type substitutes a deployment artifact with the same name and type of the Node Type derived from.
- Policies: The set of policies of this Node Type consists of the set union of the policies defined by the Nodes Type derived from and the policies defined by this Node Type. A policy defined by this Node Type substitutes a policy with the same name and type of the Node Type derived from.

## 4.4 Example

The following example defines the Node Type "Project". It is defined in a Service Template "myService" within the target namespace "http://www.ibm.com/sample". Thus, by importing the corresponding namespace in another Service Template, the Project Node Type is available for use in the other Service Template.

```
<ServiceTemplate id="myService" name="My Service"
  targetNamespace="http://www.ibm.com/sample">

  <NodeTypes>

    <NodeType id="Project" name="My Project">

      <documentation xml:lang="EN">
        A reusable definition of a node type supporting
        the creation of new projects.
      </documentation>

      <NodeTypeProperties element="ProjectProperties"/>

      <InstanceStates>
        <InstanceState state="www.my.com/active"/>
        <InstanceState state="www.my.com/onHalt"/>
      </InstanceStates>
    </NodeType>
  </NodeTypes>
</ServiceTemplate>
```

```

610     </InstanceStates>
611
612     <Interfaces>
613         <Interface>
614             <Operation name="CreateProject">
615                 <InputParameters>
616                     <InputParamter name="ProjectName"
617                                     type="string"/>
618                     <InputParamter name="Owner"
619                                     type="string"/>
620                     <InputParamter name="AccountID"
621                                     type="string"/>
622                 </InputParameters>
623                 <Implementations>
624                     <Implementation>
625                         ...
626                     </Implementation>
627                 </Implementations>
628             </Operation>
629         </Interface>
630     </Interfaces>
631
632 </NodeType>
633
634 </NodeTypes>
635
636 </ServiceTemplate>

```

637 The Node Type "Project" has three Node Type Properties defined as an XML element in the `Types`  
 638 element definition of the Service Template document: `Owner`, `ProjectName` and `AccountID` which are all  
 639 of type "string". An instance of the Node Type "Project" could be "active" (more precise in state  
 640 `www.my.com/active`) or "on hold" (more precise in state `www.my.com/onHold`). A single Interface is  
 641 defined for this Node Type, and this Interface is defined by an Operation, i.e. its actual implementation is  
 642 defined by the definition of the Operation. The Operation has the name `CreateProject` and two Input  
 643 Parameters (exploiting the default value "yes" of the attribute `required` of the `InputParameter`  
 644 element). The names of these two Input Parameters are `ProjectName` and `AccountID`, both of type  
 645 "string".

---

## 5 Relationship Types

This chapter specifies how *Relationship Types* are defined. A Relationship Type is a reusable entity that defines the type of one or more Relationship Templates between Node Templates. A Relationship Type can define observable properties via *Relationship Type Properties*. Furthermore, it defines the potential states an instance of it might reveal at runtime.

### 5.1 Syntax

```
1 <RelationshipTypes>
2
3   <RelationshipType id="ID"
4       name="string"?
5       semantics="anyURI"
6       cascadingDeletion="yes|no"?>+
7
8   <RelationshipTypeProperties element="QName"?
9       type="QName"?/>?
10
11   <InstanceStates?>
12     <InstanceState state="anyURI">+
13   </InstanceStates>
14
15 </RelationshipType>
16
17 </RelationshipTypes>
```

### 5.2 Properties

The `RelationshipType` element has the following properties:

- `id`: This attribute specifies the identifier of the Relationship Type. The identifier of the Relationship Type MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Relationship Type.
- `semantics`: The meaning or expected behavior of an instance of this Relationship Type.
- `cascadingDeletion`: If set to “yes” the target of an instance of a Relationship Template of this RelationshipType is automatically deleted when the source of the instance of the Relationship Template is deleted.

The `RelationshipTypeProperties` element has the following properties:

- `element`: The QName value of this attribute refers to an XML element defining the structure of the Relationship Type Properties.
- `type`: The QName value of this attribute refers to an XML (complex) type defining the structure of the Relationship Type Properties.

Either the `element` attribute or the `type` attribute MUST be specified, but not both.

The `InstanceStates` element has the following properties:

- `InstanceState`: specifies a potential state.

The `InstanceState` element has the following properties:

- `state`: a URI that represents a potential state.

## 5.3 Example

The following example defines the Relationship Type “processDeployedOn”. The meaning of this Relationship Type is that “a process is deployed on a hosting environment” (indicated by the URI value of the `semantics` attribute). When the source of an instance of a Relationship Template referring to this Relationship Type is deleted, its target is automatically deleted as well. The Relationship Type has Relationship Type Properties defined in the `Types` section of the same Service Template document as the `ProcessDeployedOnProperties` element. The states an instance of this Relationship Type can be in are also listed.

```
<RelationshipTypes>
  <RelationshipType id="processDeployedOn"
    name="Process is deployed on"
    semantics="www.my.com/RelSemantics/procDeployedOn"
    cascadingDeletion="yes">
    <RelationshipTypeProperties element="ProcessDeployedOnProperties"/>
    <InstanceStates>
      <InstanceState state="www.my.com/successfullyDeployed"/>
      <InstanceState state="www.my.com/failed"/>
    </InstanceStates>
  </RelationshipType>
</RelationshipTypes>
```

## 6 Topology Template

This chapter specifies how *Topology Templates* are defined. A Topology Template defines the overall structure of an IT service, i.e. the components it consists of, the relations between those components, as well as grouping of components. The components of a service are referred to as *Node Templates*, the relations between the components are referred to as *Relationship Templates*, and groupings are referred to as *Group Templates*.

### 6.1 Syntax

```
1  <TopologyTemplate id="ID"
2      name="string"?>
3
4  (
5      <NodeTemplate id="ID"
6          name="string"?
7          nodeType="QName"
8          minInstances="int"?
9          maxInstances="int|string"?>
10
11      <PropertyDefaults>?
12          XML fragment
13      </PropertyDefaults>
14
15      <PropertyConstraints>?
16
17          <PropertyConstraint property="string"
18              constraintType="anyURI">+
19              constraint?
20          </PropertyConstraint>
21
22      </PropertyConstraints>
23
24      <Policies>?
25          <Policy name="string" type="anyURI">+
26              policy specific content
27          </Policy>
28      </Policies>
29
30      <EnvironmentConstraints>?
31          <EnvironmentConstraint constraintType="anyURI">+
32              constraint type specific content?
33          </EnvironmentConstraint>
34      </EnvironmentConstraints>
35
36      <DeploymentArtifacts>?
37          <DeploymentArtifact name="string" type="anyURI">+
38              artifact specific content
39          </DeploymentArtifact>
40      </DeploymentArtifacts>
41
42      </NodeTemplate>
43  |
44      <RelationshipTemplate id="ID"
45          name="string"?
```



```

765 46             relationshipType="QName">
766 47
767 48         <SourceElement id="IDREF"/>
768 49
769 50     ( <TargetElement id="IDREF"/>
770 51     |
771 52         <TargetElementReference id="QName"/>
772 53     )
773 54
774 55     <PropertyDefaults>?
775 56         XML fragment
776 57     </PropertyDefaults>
777 58
778 59     <PropertyConstraints>?
779 60
780 61         <PropertyConstraint property="string"
781 62             constraintType="anyURI">+
782 63             constraint?
783 64         </PropertyConstraint>
784 65
785 66     </PropertyConstraints>
786 67
787 68     <RelationshipConstraints>?
788 69
789 70         <RelationshipConstraint constraintType="anyURI">+
790 71             constraint?
791 72         </RelationshipConstraint>
792 73
793 74     </RelationshipConstraints>
794 75
795 76 </RelationshipTemplate>
796 77 |
797 78 <GroupTemplate id="ID"
798 79     name="string"?
799 80     minInstances="int"?
800 81     maxInstances="int|string"?>
801 82
802 83     (
803 84         <NodeTemplate ... />
804 85     |
805 86         <RelationshipTemplate ... />
806 87     |
807 88         <GroupTemplate ... />
808 89     )+
809 90
810 91     <Policies>?
811 92         <Policy name="string" type="anyURI">+
812 93             policy specific content
813 94         </Policy>
814 95     </Policies>
815 96
816 97 </GroupTemplate>
817 98 )+
818 99
819 100 </TopologyTemplate>

```

## 6.2 Properties

The `TopologyTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Topology Template. The identifier of the Topology Template MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Topology Template.
- `NodeTemplate`: This is a kind of a component making up the IT service.
- `RelationshipTemplate`: This is a kind of relationship between the components (nodes or groups) of the service.
- `GroupTemplate`: This is a grouping of node templates, relationship templates, or (nested) group templates within the Topology Templates to express a special association between the grouped elements.

A Topology Template can contain any number of Node Templates, Relationship Templates, or Group Templates (i.e. “elements”). For each specified Relationship Template (either defined as a direct child of the Topology Template or within a Group Template) the source element and target element MUST be specified in the Topology Template except for target elements that are referenced (via a target element reference).

The `NodeTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Node Template. The identifier of the Node Template MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Node Template.
- `nodeType`: The QName value of this attribute refers to the Node Type providing the type of the Node Template.
- `minInstances`: This integer attribute specifies the minimum number of instances to be created when instantiating the Node Template. The default value of this attribute is 1. The value of `minInstances` MUST NOT be less than 0.
- `maxInstances`: This attribute specifies the maximum number of instances that can be created when instantiating the Node Template. The default value of this attribute is 1. If the string is set to “unbounded”, an unbounded number of instances can be created. The value of `maxInstances` MUST be 1 or greater and MUST NOT be less than the value specified for `minInstances`.
- `PropertyDefaults`: Specifies initial values for one or more of the Node Type Properties of the Node Type providing the property definitions in the concrete context of the Node Template.  
  
The initial values are specified by providing an instance document of the XML schema of the corresponding Node Type Properties. This instance document considers the inheritance structure deduced by the `DerivedFrom` property of the Node Type referenced by the `nodeType` attribute of the Node Template.  
  
The instance document of the XML schema might not validate against the existence constraints of the corresponding schema: not all node type properties might have an initial value assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the Property Defaults element. Once the defined Node Template has been instantiated, any XML representation of the Node Type properties MUST validate according to the associated XML schema definition.
- `PropertyConstraints`: Specifies constraints on the use of one or more of the Node Type Properties of the Node Type providing the property definitions for the Node Template.

Each constraint is specified by means of a separate nested `PropertyConstraint` element. This element contains the actual encoding of the constraint.

- **Policies:** Specifies policies of the Node Template. Each policy is specified by means of a separate nested `Policy` element. This element contains the actual policy specific content of the policy.

Note, that a policy specified in the Node Template overrides any policy of the same name and type that might be specified with the Node Type of this Node Template.

Any policies of the Node Type that are not overridden are combined with the policies of the Node Template.

- **EnvironmentConstraints:** The nested `EnvironmentConstraint` elements of the Node Template under definition constrain the runtime environment for the corresponding component of a service. For example, constraints on network security settings of the hosting environment or requirements on the existence of certain resources might be defined within the Environment Constraints definition of a Node Template.
- **DeploymentArtifacts:** This element specifies the deployment artifacts relevant for the Node Template under definition.

Its nested `DeploymentArtifact` elements specify details about individual deployment artifacts. The `name` attribute of a `DeploymentArtifact` element specifies the name of the artifact. Uniqueness of the name within the scope of the encompassing Node Template SHOULD be guaranteed by the definition. The `type` attribute of a `DeploymentArtifact` element specifies the type of the deployment artifact definition that is related to the Node Template, i.e. the attribute gives a hint how to interpret the body of the `DeploymentArtifact` element. The body of this element contains the type-specific content.

For example, if the `type` attribute contains the value `http://docs.oasis-open.org/tosca/ns/2011/12/deploymentArtifacts/ovfRef`, the body will contain an XML fragment with a reference to an OVF package and a mapping between service template data and elements of the respective OVF envelope.

Note, that a Deployment Artifact specified with the Node Template under definition overrides any Deployment Artifact of the same name and the same type specified with the Node Type given as value of the `nodeType` attribute of the Node Template under definition.

Otherwise, the Deployment Artifacts of the Node Type given as value of the `nodeType` attribute of the Node Template under definition and the Deployment Artifacts defined with the Node Template are combined.

The `PropertyConstraint` element has the following properties:

- **property:** The string value of this property is an XPath expression pointing to the property within the Node Type Properties document that is constrained within the context of the Node Template. More than one constraint MUST NOT be defined for each property.
- **constraintType:** The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content.

For example, a constraint type of `http://www.example.com/PropertyConstraints/unique` could denote that the reference property of the node template under definition has to be unique within a

912 certain scope. The constraint type specific content of the respective `PropertyConstraint`  
913 element could then define the actual scope in which uniqueness has to be ensured in more detail.

914 The `Policy` element has the following properties:

- 915 • `type`: This attribute specifies the kind of policy (e.g. management practice) supported by an  
916 instance of the Node Type containing this element.
- 917 • `name`: This attribute defines the name of the policy. The name **MUST** be unique within a given  
918 Node Type containing the `Policy` element.

919 The `EnvironmentConstraint` element has the following properties:

- 920 • `constraintType`: The constraint type is specified by means of a URI, which defines both the  
921 semantic meaning of the constraint as well as the format of the constraint content.

922 The `RelationshipTemplate` element has the following properties:

- 923 • `id`: This attribute specifies the identifier of the Relationship Template. The identifier of the  
924 Relationship Template **MUST** be unique within the target namespace.
- 925 • `name`: This optional attribute specifies the name of the Relationship Template.
- 926 • `relationshipType`: The QName value of this property refers to the Relationship Type  
927 providing the type of the Relationship Template.
- 928 • `SourceElement`: The `id` attribute of this element references a Node Template or Group  
929 Template within the same Service Template document that is the source of the Relationship  
930 Template.
- 931 • `TargetElement`: The `id` attribute of this element references a Node Template or Group  
932 Template within the same Service Template document that is the target of the Relationship  
933 Template.
- 934 • `TargetElementReference`: The `id` attribute of this element refers by QName to an imported  
935 Node Template or Group Template that is the target of the Relationship Template. The  
936 referenced Node Template or Group Template will typically be the root node or root group of the  
937 corresponding Topology Template. In some cases a non-root Node Template or non-root Group  
938 Template might be referenced to support access to particular resources from a larger service, for  
939 example. Either `TargetElement` or `TargetElementReference` **MUST** be specified but not  
940 both.
- 941 • `PropertyDefaults`: Specifies initial values for one or more of the Relationship Type Properties  
942 of the Relationship Type providing the property definitions in the concrete context of the  
943 Relationship Template.  
944 The initial values are specified by providing an instance document of the XML schema of the  
945 corresponding Relationship Type Properties.  
946 The instance document of the XML schema might not validate against the existence constraints  
947 of the corresponding schema: not all relationship type properties might have an initial value  
948 assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the  
949 Property Defaults element. Once the defined Relationship Template has been instantiated, any  
950 XML representation of the Relationship Type properties **MUST** validate according to the  
951 associated XML schema definition.
- 952 • `PropertyConstraints`: Specifies constraints on the use of one or more of the Relationship  
953 Type Properties of the Relationship Type providing the property definitions for the Relationship  
954 Template.  
955 Each constraint is specified by means of a separate nested `PropertyConstraint` element.  
956 This element contains the actual encoding of the constraint.

- **RelationshipConstraints:** Specifies constraints on the use of the relationship. Each constraint is specified by means of a separate nested `RelationshipConstraint` element. This element can contain the actual encoding of the constraint, or its `constraintType` attribute already denotes the constraint itself. The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of any content.

The `GroupTemplate` element has the following properties:

- **id:** This attribute specifies the identifier of the Group Template. The identifier of the Group Template **MUST** be unique within the target namespace.
- **name:** This optional attribute specifies the name of the Group Template.
- **minInstances:** This integer attribute specifies the minimum number of instances to be created when instantiating the Group Template. The default value of this attribute is 1. The value of `minInstances` **MUST NOT** be less than 0.
- **maxInstances:** This attribute specifies the maximum number of instances that can be created when instantiating the Group Template. The default value of this attribute is 1. If the string is set to “unbounded”, an unbounded number of instances can be created. The value of `maxInstances` **MUST** be 1 or greater and **MUST NOT** be less than the value specified for `minInstances`.
- **NodeTemplate:** This is a node template contained within, or grouped by the Group Template.
- **RelationshipTemplate:** This is a relationship template contained within, or grouped by the Group.
- **GroupTemplate:** This is a Group Template of a nested group contained within, or grouped by the Group Template.
- **Policies:** Specifies policies of the Group Template. Each policy is specified by means of a separate nested `Policy` element. This element contains the actual policy specific content of the policy.

## 6.3 Example

The following Service Template defines a Topology Template in-place. The corresponding Topology Template contains two Node Templates called “MyApplication” and “MyAppServer”. These Node Templates have the node types “Application” and “ApplicationServer”, respectively, the definitions of which are imported by the `Import` element. The Node Template “MyApplication” is instantiated exactly once. Two of its Node Type Properties are initialized by a corresponding `PropertyDefaults` element. The Node Template “MyAppServer” can be instantiated as many times as needed. The “MyApplication” Node Template is connected with the “MyAppServer” Node Template via the Relationship Template named “MyDeploymentRelationship”; the behavior and semantics of the Relationship Template is defined in the Relationship Type “deployedOn” in the same Service Template document, saying that “MyApplication” is deployed on “MyAppServer”. When instantiating the “SampleApplication” Topology Template, instances of “MyApplication” and “MyAppServer” are related by means of corresponding instances of “MyDeploymentRelationship”.

```
<ServiceTemplate id="myService"
  name="My Service"
  targetNamespace="http://www.ibm.com/sample"
  xmlns:abc="http://www.ibm.com/sample">

  <Import namespace="http://www.ibm.com/sample"
    importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>

  <TopologyTemplate id="SampleApplication">
```

```

1004 <NodeTemplate id="MyApplication"
1005         name="My Application"
1006         nodeType="abc:Application">
1007     <PropertyDefaults>
1008         <ApplicationProperties>
1009             <Owner>Frank</Owner>
1010             <InstanceName>Thomas' favorite application</InstanceName>
1011         </ApplicationProperties>
1012     </PropertyDefaults>
1013 </NodeTemplate/>
1014
1015 <NodeTemplate id="MyAppServer"
1016         name="My Application Server"
1017         nodeType="abc:ApplicationServer"
1018         minInstances="0"
1019         maxInstances="unbounded"/>
1020
1021 <RelationshipTemplate id="MyDeploymentRelationship"
1022         relationshipType="deployedOn">
1023     <SourceElement id="MyApplication"/>
1024     <TargetElement id="MyAppServer"/>
1025 </RelationshipTemplate>
1026
1027 </TopologyTemplate>
1028
1029 </ServiceTemplate>

```

---

## 7 Plans

The operational management behavior of a Service Template is invoked by means of orchestration plans, or more simply, *Plans*. Plans consist of individual steps (aka tasks or activities) to be performed and the definition of the potential order of these steps. The execution of a step can be performed by one of the functions offered via the interfaces of a Node Template, by invoking operations of a Service Template API, or by invoking other operations being required in the context of a specific service. Plans are classified by a type, and the following two plan types are defined as part of the TOSCA specification. *Build plans* specify how instances of their associated Service Templates are made, and *termination plans* specify how an instance of a Service Template is removed from the environment. Other plan types for managing existing service instances throughout their life time are termed *modification plans*, and it is expected that such plan types will be defined subsequently by authors of service templates and domain expert groups.

### 7.1 Syntax

```
1  <Plans>
2
3    <Plan id="ID"
4        name="string"?
5        planType="anyURI"
6        languageUsed="anyURI">+
7
8        <PreCondition expressionLanguage="anyURI">?
9            condition
10        </PreCondition>
11
12        ( <PlanModel>
13            actual plan
14        </PlanModel>
15        |
16        <PlanModelReference reference="anyURI"/>
17        )
18
19    </Plan>
20
21 </Plans>
```

### 7.2 Properties

The `Plans` element contains one or more `Plan` elements which have the following properties:

- `id`: This attribute specifies the identifier of the Plan. The identifier of the Plan MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Plan.
- `planType`: The value of the attribute specifies the type of the plan as an indication on what the effect of executing the plan on a service will have. The plan type is specified by means of a URI, allowing for an extensibility mechanism for authors of service templates to define new plan types over time.

The following plan types are defined as part of the TOSCA specification.

1074                   ○ <http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan> - This URI defines the  
1075 *build plan* plan type for plans used to initially create a new instance of a service from a  
1076 Service Template.

1077                   ○ <http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan> - This URI  
1078 defines the *termination plan* plan type for plans used to terminate the existence of a  
1079 service instance.

1080 Note that all other plan types for managing service instances throughout their life time will be  
1081 considered and referred to as *modification plans* in general.

1082       • `languageUsed`: This attribute denotes the process modeling language (or metamodel) used to  
1083 specify the plan. For example, "<http://www.omg.org/spec/BPMN/2.0/>" would specify that BPMN  
1084 2.0 has been used to model the plan.

1085       • `PreCondition`: This optional element specifies a condition that needs to be satisfied in order for  
1086 the plan to be executed. The `expressionLanguage` attribute of this element specifies the  
1087 expression language the nested condition is provided in.

1088 Typically, the precondition will be an expression in the instance state attribute of some of the  
1089 node templates or relationship templates of the topology template. It will be evaluated based on  
1090 the actual values of the corresponding attributes at the time the plan is requested to be executed.  
1091 Note, that any other kind of pre-condition is allowed.

1092       • `PlanModel`: This property contains the actual model content.

1093       • `PlanModelReference`: This property points to the model content. Its reference attribute  
1094 contains a URI of the model of the plan.

1095  
1096 An instance of the `Plan` element MUST either contain the actual plan as instance of the  
1097 `PlanModel` element, or point to the model via the `PlanModelReference` element.

## 1098 7.3 Use of Process Modeling Languages

1099 TOSCA does not specify a separate metamodel for defining plans. Instead, it is assumed that a process  
1100 modelling language (a.k.a. metamodel) like BPEL [BPEL 2.0] or BPMN [BPMN 2.0] is used to define  
1101 plans. The specification favours the use of BPMN for modeling plans.

## 1102 7.4 Example

1103 The following defines two Plans, one Plan for creating a new instance of the "SampleApplication"  
1104 Topology Template (the plan is named "DeployApplication"), and one Plan for removing instances of  
1105 "SampleApplication". The Plan "DeployApplication" is a build plan specified in BPMN; the process model  
1106 is immediately included in the Plan Model (note that the BPMN model is incomplete but used to show the  
1107 mechanism of the `PlanModel` element). The Plan can only run when the PreCondition "Run only if  
1108 funding is available" is satisfied. The Plan "RemoveApplication" is a termination plan specified in BPEL;  
1109 the corresponding BPEL definition is defined elsewhere and only referenced by the  
1110 `PlanModelReference` element.

```
1111 <Plans>
1112
1113   <Plan id="DeployApplication"
1114     name="Sample Application Build Plan"
1115     planType=
1116       "http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
1117     languageUsed="http://www.omg.org/spec/BPMN/2.0/">
1118
1119     <PreCondition expressionLanguage="www.my.com/text">?
1120       Run only if funding is available
```



```

1121     </PreCondition>
1122
1123     <PlanModel>
1124         <process name="DeployNewApplication" id="p1">
1125             <documentation>This process deploys a new instance of the
1126                 sample application.
1127             </documentation>
1128
1129             <task id="t1" name="CreateAccount"/>
1130
1131             <task id="t2" name="AcquireNetworkAddresses"
1132                 isSequential="false"
1133                 loopDataInput="t2Input.LoopCounter"/>
1134             <documentation>Assumption: t2 gets data of type "input"
1135                 as input and this data has a field names "LoopCounter"
1136                 that contains the actual multiplicity of the task.
1137             </documentation>
1138
1139             <task id="t3" name="DeployApplicationServer"
1140                 isSequential="false"
1141                 loopDataInput="t3Input.LoopCounter"/>
1142
1143             <task id="t4" name="DeployApplication"
1144                 isSequential="false"
1145                 loopDataInput="t4Input.LoopCounter"/>
1146
1147             <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
1148             <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
1149             <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
1150         </process>
1151     </PlanModel>
1152 </Plan>
1153
1154 <Plan id="RemoveApplication"
1155     planType="http://docs.oasis-
1156     open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
1157     languageUsed=
1158         "http://docs.oasis-open.org/wsbpel/2.0/process/executable">
1159     <PlanModelReference reference="prj:RemoveApp"/>
1160 </Plan>
1161
1162 </Plans>

```

---

## 8 Security Considerations

1163

1164

1165

TOSCA does not mandate the use of any specific mechanism or technology for client authentication.  
However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the infrastructure.

---

1166 **9 Conformance**

1167 **This section is to be done.**

---

## Appendix A. Portability and Interoperability Considerations

This section illustrates the portability and interoperability aspects addressed by Service Templates:

Portability - The ability to take Service Templates created in one vendor's environment and use them in another vendor's environment.

Interoperability - The capability for multiple components (e.g. a task of a plan and the definition of a topology node) to interact using well-defined messages and protocols. This enables combining components from different vendors allowing seamless management of services.

Portability demands support of TOSCA artifacts.

## Appendix B. Complete TOSCA Grammar

```
1177
1178 1 <ServiceTemplate id="ID"
1179 2     name="string"?
1180 3     targetNamespace="anyURI">
1181 4
1182 5     <Extensions>?
1183 6         <Extension namespace="anyURI"
1184 7             mustUnderstand="yes|no"?/>+
1185 8     </Extensions>
1186 9
1187 10    <Import namespace="anyURI"?
1188 11        location="anyURI"?
1189 12        importType="anyURI"/>*
1190 13
1191 14    <Types>?
1192 15        <xs:schema .../>*
1193 16    </Types>
1194 17
1195 18    (
1196 19        <TopologyTemplateReference reference="QName"/>
1197 20        |
1198 21        <TopologyTemplate id="ID"
1199 22            name="string"?>
1200 23
1201 24            (
1202 25                <NodeTemplate id="ID"
1203 26                    name="string"?
1204 27                    nodeType="QName"
1205 28                    minInstances="int"?
1206 29                    maxInstances="int|string"?>
1207 30
1208 31                    <PropertyDefaults>?
1209 32                        XML fragment
1210 33                    </PropertyDefaults>
1211 34
1212 35                    <PropertyConstraints>?
1213 36
1214 37                        <PropertyConstraint property="string"
1215 38                            constraintType="anyURI">+
1216 39                            constraint?
1217 40                        </PropertyConstraint>
1218 41
1219 42                    </PropertyConstraints>
1220 43
1221 44                    <Policies>?
1222 45                        <Policy name="string" type="anyURI">+
1223 46                            policy specific content
1224 47                        </Policy>
1225 48                    </Policies>
1226 49
1227 50                    <EnvironmentConstraints>?
1228 51                        <EnvironmentConstraint constraintType="anyURI">+
1229 52                            constraint type specific content?
1230 53                    </EnvironmentConstraint>
```

```

1231 54      </EnvironmentConstraints>
1232 55
1233 56      <DeploymentArtifacts>?
1234 57          <DeploymentArtifact name="string" type="anyURI">+
1235 58              artifact specific content
1236 59          </DeploymentArtifact>
1237 60      </DeploymentArtifacts>
1238 61
1239 62  </NodeTemplate>
1240 63  |
1241 64  <RelationshipTemplate id="ID"
1242 65      name="string"?
1243 66      relationshipType="QName">+
1244 67
1245 68      <SourceElement id="IDREF"/>
1246 69
1247 70      ( <TargetElement id="IDREF"/>
1248 71          |
1249 72          <TargetElementReference id="QName"/>
1250 73      )
1251 74
1252 75      <PropertyDefaults>?
1253 76          XML fragment
1254 77      </PropertyDefaults>
1255 78
1256 79      <PropertyConstraints>?
1257 80
1258 81          <PropertyConstraint property="string"
1259 82              constraintType="anyURI">+
1260 83              constraint?
1261 84          </PropertyConstraint>
1262 85
1263 86      </PropertyConstraints>
1264 87
1265 88      <RelationshipConstraints>?
1266 89
1267 90          <RelationshipConstraint constraintType="anyURI">+
1268 91              constraint?
1269 92          </RelationshipConstraint>
1270 93
1271 94      </RelationshipConstraints>
1272 95
1273 96  </RelationshipTemplate>
1274 97  |
1275 98  <GroupTemplate id="ID"
1276 99      name="string"?
1277 100      minInstances="int"?
1278 101      maxInstances="int|string"?>
1279 102
1280 103      (
1281 104          <NodeTemplate ... />
1282 105          |
1283 106          <RelationshipTemplate ... />
1284 107          |
1285 108          <GroupTemplate ... />
1286 109      )+
1287 110
1288 111      <Policies>?

```

```

1289 112         <Policy name="string" type="anyURI">+
1290 113             policy specific content
1291 114         </Policy>
1292 115     </Policies>
1293 116
1294 117     </GroupTemplate>
1295 118 )+
1296 119
1297 120 </TopologyTemplate>
1298 121 )?
1299 122
1300 123 <NodeTypes>?
1301 124
1302 125     <NodeType id="ID"
1303 126         name="string"?>+
1304 127
1305 128         <NodeTypeProperties element="QName"?
1306 129             type="QName"?/>?
1307 130
1308 131         <DerivedFrom nodeTypeRef="QName"/>?
1309 132
1310 133     <InstanceStates>?
1311 134         <InstanceState state="anyURI">+
1312 135     </InstanceStates>
1313 136
1314 137     <Interfaces>?
1315 138
1316 139     <Interface>+
1317 140
1318 141         (
1319 142         <WSDL portType="QName"
1320 143             operation="NCName"?>+
1321 144         |
1322 145         <REST method="GET | PUT | POST | DELETE"
1323 146             requestURI="anyURI"
1324 147             requestPayload="QName"?
1325 148             responsePayload="QName"?>+
1326 149         |
1327 150         <Operation name="NCame">+
1328 151
1329 152         <InputParameters>?
1330 153
1331 154             <InputParamter name="string"
1332 155                 type="string"
1333 156                 required="yes|no">+
1334 157
1335 158         </InputParameters>
1336 159
1337 160         <OutputParameters>?
1338 161
1339 162             <OutputParamter name="string"
1340 163                 type="string"
1341 164                 required="yes|no">+
1342 165
1343 166         </OutputParameters>
1344 167
1345 168         <Implementations>
1346 169

```

```

1347 170         <Implementation implementationID="anyURI"?
1348 171             language="anyURI"?>+
1349 172         (
1350 173             <ImplementationProper>?
1351 174                 code
1352 175             </ImplementationProper>
1353 176             |
1354 177             <ImplementationReference ref="anyURI"/>?
1355 178         )
1356 179     </Implementation>
1357 180
1358 181     </Implementations>
1359 182 </Operation>
1360 183 )
1361 184
1362 185 </Interface>
1363 186
1364 187 </Interfaces>
1365 188
1366 189 <DeploymentArtifacts>?
1367 190     <DeploymentArtifact name="string" type="anyURI">+
1368 191         artifact specific content
1369 192     </DeploymentArtifact>
1370 193 </DeploymentArtifacts>
1371 194
1372 195
1373 196 <Policies>?
1374 197
1375 198     <Policy name="string" type="anyURI">+
1376 199         policy specific content
1377 200     </Policy>
1378 201
1379 202 </Policies>
1380 203
1381 204 </NodeType>
1382 205
1383 206 </NodeTypes>
1384 207
1385 208 <RelationshipTypes>?
1386 209
1387 210     <RelationshipType id="ID"
1388 211         name="string"?
1389 212         semantics="anyURI"
1390 213         cascadingDeletion="yes|no"?>+
1391 214
1392 215     <RelationshipTypeProperties element="QName"?
1393 216         type="QName"?/>?
1394 217
1395 218     <InstanceStates>?
1396 219         <InstanceState state="anyURI">+
1397 220     </InstanceStates>
1398 221
1399 222 </RelationshipType>
1400 223
1401 224 </RelationshipTypes>
1402 225
1403 226 <Plans>?
1404 227

```



```

1405 228     <Plan id="ID"
1406 229         name="string"?
1407 230         planType="anyURI"
1408 231         languageUsed="anyURI">+
1409 232
1410 233     <PreCondition expressionLanguage="anyURI">?
1411 234         condition
1412 235     </PreCondition>
1413 236
1414 237     ( <PlanModel>
1415 238         actual plan
1416 239     </PlanModel>
1417 240     |
1418 241     <PlanModelReference reference="anyURI"/>
1419 242     )
1420 243
1421 244     </Plan>
1422 245
1423 246     </Plans>
1424 247
1425 248 </ServiceTemplate>

```

---

## Appendix C. TOSCA Schema

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
3   elementFormDefault="qualified" attributeFormDefault="unqualified"
4   xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
5   xmlns:xs="http://www.w3.org/2001/XMLSchema">
6
7   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
8     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
9
10  <xs:element name="documentation" type="tDocumentation"/>
11  <xs:complexType name="tDocumentation" mixed="true">
12    <xs:sequence>
13      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
14    </xs:sequence>
15    <xs:attribute name="source" type="xs:anyURI"/>
16    <xs:attribute ref="xml:lang"/>
17  </xs:complexType>
18
19  <xs:complexType name="tExtensibleElements">
20    <xs:sequence>
21      <xs:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
22      <xs:any namespace="##other" processContents="lax" minOccurs="0"
23        maxOccurs="unbounded"/>
24    </xs:sequence>
25    <xs:anyAttribute namespace="##other" processContents="lax"/>
26  </xs:complexType>
27
28  <xs:complexType name="tImport">
29    <xs:complexContent>
30      <xs:extension base="tExtensibleElements">
31        <xs:attribute name="namespace" type="xs:anyURI"/>
32        <xs:attribute name="location" type="xs:anyURI"/>
33        <xs:attribute name="importType" type="importedURI" use="required"/>
34      </xs:extension>
35    </xs:complexContent>
36  </xs:complexType>
37
38  <xs:element name="ServiceTemplate">
39    <xs:complexType>
40      <xs:complexContent>
41        <xs:extension base="tServiceTemplate"/>
42      </xs:complexContent>
43    </xs:complexType>
44  </xs:element>
45
46  <xs:complexType name="tServiceTemplate">
47    <xs:complexContent>
48      <xs:extension base="tExtensibleElements">
49        <xs:sequence>
50          <xs:element name="Import" type="tImport" minOccurs="0"
51            maxOccurs="unbounded"/>
52          <xs:element name="Types" minOccurs="0">
53            <xs:complexType>
```

```

1480 54      <xs:sequence>
1481 55      <xs:any namespace="##other" processContents="lax" minOccurs="0"
1482 56      maxOccurs="unbounded"/>
1483 57      </xs:sequence>
1484 58      </xs:complexType>
1485 59      </xs:element>
1486 60      <xs:element name="Extensions" minOccurs="0">
1487 61      <xs:complexType>
1488 62      <xs:sequence>
1489 63      <xs:element name="Extension" type="tExtension"
1490 64      maxOccurs="unbounded"/>
1491 65      </xs:sequence>
1492 66      </xs:complexType>
1493 67      </xs:element>
1494 68      <xs:choice minOccurs="0">
1495 69      <xs:element name="TopologyTemplateReference">
1496 70      <xs:complexType>
1497 71      <xs:attribute name="reference" type="xs:QName"/>
1498 72      </xs:complexType>
1499 73      </xs:element>
1500 74      <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1501 75      </xs:choice>
1502 76      <xs:element name="NodeTypes" type="tNodeTypes" minOccurs="0"/>
1503 77      <xs:element name="RelationshipTypes" type="tRelationshipTypes"
1504 78      minOccurs="0"/>
1505 79      <xs:element name="Plans" type="tPlans" minOccurs="0"/>
1506 80      </xs:sequence>
1507 81      <xs:attribute name="id" type="xs:ID" use="required"/>
1508 82      <xs:attribute name="name" type="xs:string" use="optional"/>
1509 83      <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1510 84      </xs:extension>
1511 85      </xs:complexContent>
1512 86      </xs:complexType>
1513 87
1514 88      <xs:complexType name="tDeploymentArtifact">
1515 89      <xs:complexContent>
1516 90      <xs:extension base="tExtensibleElements">
1517 91      <xs:attribute name="name" type="xs:string" use="required"/>
1518 92      <xs:attribute name="type" type="xs:anyURI" use="required"/>
1519 93      </xs:extension>
1520 94      </xs:complexContent>
1521 95      </xs:complexType>
1522 96
1523 97      <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1524 98      <xs:complexType name="tNodeTemplate">
1525 99      <xs:complexContent>
1526 100      <xs:extension base="tExtensibleElements">
1527 101      <xs:sequence>
1528 102      <xs:element name="PropertyDefaults" minOccurs="0">
1529 103      <xs:complexType>
1530 104      <xs:sequence>
1531 105      <xs:any namespace="##other" processContents="lax"/>
1532 106      </xs:sequence>
1533 107      </xs:complexType>
1534 108      </xs:element>
1535 109      <xs:element name="PropertyConstraints" minOccurs="0">
1536 110      <xs:complexType>
1537 111      <xs:sequence>

```

```

1538 112      <xs:element name="PropertyConstraint"
1539 113      type="tPropertyConstraint" maxOccurs="unbounded"/>
1540 114    </xs:sequence>
1541 115  </xs:complexType>
1542 116 </xs:element>
1543 117 <xs:element name="Policies" minOccurs="0">
1544 118   <xs:complexType>
1545 119     <xs:sequence>
1546 120       <xs:element name="Policy" type="tPolicy"
1547 121       maxOccurs="unbounded"/>
1548 122     </xs:sequence>
1549 123   </xs:complexType>
1550 124 </xs:element>
1551 125 <xs:element name="DeploymentArtifacts" minOccurs="0">
1552 126   <xs:complexType>
1553 127     <xs:sequence>
1554 128       <xs:element name="DeploymentArtifact"
1555 129       type="tDeploymentArtifact" maxOccurs="unbounded"/>
1556 130     </xs:sequence>
1557 131   </xs:complexType>
1558 132 </xs:element>
1559 133 <xs:element name="EnvironmentConstraints" minOccurs="0">
1560 134   <xs:complexType>
1561 135     <xs:sequence>
1562 136       <xs:element name="EnvironmentConstraint"
1563 137       type="tEnvironmentConstraint" maxOccurs="unbounded"/>
1564 138     </xs:sequence>
1565 139   </xs:complexType>
1566 140 </xs:element>
1567 141 </xs:sequence>
1568 142 <xs:attribute name="id" type="xs:ID" use="required"/>
1569 143 <xs:attribute name="name" type="xs:string" use="optional"/>
1570 144 <xs:attribute name="nodeType" type="xs:QName" use="required"/>
1571 145 <xs:attribute name="minInstances" type="xs:int" use="optional"
1572 146   default="1"/>
1573 147 <xs:attribute name="maxInstances" use="optional" default="1">
1574 148   <xs:simpleType>
1575 149     <xs:union>
1576 150       <xs:simpleType>
1577 151         <xs:restriction base="xs:nonNegativeInteger">
1578 152           <xs:pattern value="([1-9]+[0-9]*)"/>
1579 153         </xs:restriction>
1580 154       </xs:simpleType>
1581 155       <xs:simpleType>
1582 156         <xs:restriction base="xs:string">
1583 157           <xs:enumeration value="unbounded"/>
1584 158         </xs:restriction>
1585 159       </xs:simpleType>
1586 160     </xs:union>
1587 161   </xs:simpleType>
1588 162 </xs:attribute>
1589 163 </xs:extension>
1590 164 </xs:complexContent>
1591 165 </xs:complexType>
1592 166
1593 167 <xs:complexType name="tPropertyConstraint">
1594 168   <xs:sequence>
1595 169     <xs:any namespace="##other" processContents="lax" minOccurs="0"/>

```

```

1596 170     </xs:sequence>
1597 171     <xs:attribute name="property" type="xs:string" use="required"/>
1598 172     <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
1599 173 </xs:complexType>
1600 174
1601 175 <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1602 176 <xs:complexType name="tTopologyTemplate">
1603 177     <xs:complexContent>
1604 178         <xs:extension base="tTopologyElementCollection"/>
1605 179     </xs:complexContent>
1606 180 </xs:complexType>
1607 181
1608 182 <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1609 183 <xs:complexType name="tGroupTemplate">
1610 184     <xs:complexContent>
1611 185         <xs:extension base="tTopologyElementCollection">
1612 186             <xs:sequence>
1613 187                 <xs:element name="Policies" minOccurs="0">
1614 188                     <xs:complexType>
1615 189                         <xs:sequence>
1616 190                             <xs:element name="Policy" type="tPolicy"
1617 191                                 maxOccurs="unbounded"/>
1618 192                         </xs:sequence>
1619 193                     </xs:complexType>
1620 194                 </xs:element>
1621 195             </xs:sequence>
1622 196             <xs:attribute name="minInstances" type="xs:int" use="optional"
1623 197                 default="1"/>
1624 198             <xs:attribute name="maxInstances" use="optional" default="1">
1625 199                 <xs:simpleType>
1626 200                     <xs:union>
1627 201                         <xs:simpleType>
1628 202                             <xs:restriction base="xs:nonNegativeInteger">
1629 203                                 <xs:pattern value="([1-9]+[0-9]*)"/>
1630 204                             </xs:restriction>
1631 205                         </xs:simpleType>
1632 206                         <xs:simpleType>
1633 207                             <xs:restriction base="xs:string">
1634 208                                 <xs:enumeration value="unbounded"/>
1635 209                             </xs:restriction>
1636 210                         </xs:simpleType>
1637 211                     </xs:union>
1638 212                 </xs:simpleType>
1639 213             </xs:attribute>
1640 214         </xs:extension>
1641 215     </xs:complexContent>
1642 216 </xs:complexType>
1643 217
1644 218 <xs:complexType name="tTopologyElementCollection">
1645 219     <xs:complexContent>
1646 220         <xs:extension base="tExtensibleElements">
1647 221             <xs:choice maxOccurs="unbounded">
1648 222                 <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1649 223                 <xs:element name="RelationshipTemplate"
1650 224                     type="tRelationshipTemplate"/>
1651 225                 <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1652 226             </xs:choice>
1653 227             <xs:attribute name="id" type="xs:ID" use="required"/>

```

```

1654 228     <xs:attribute name="name" type="xs:string" use="optional"/>
1655 229     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1656 230     </xs:extension>
1657 231     </xs:complexContent>
1658 232 </xs:complexType>
1659 233
1660 234 <xs:element name="RelationshipTypes" type="tRelationshipTypes"/>
1661 235 <xs:complexType name="tRelationshipTypes">
1662 236     <xs:sequence>
1663 237         <xs:element name="RelationshipType" type="tRelationshipType"
1664 238             maxOccurs="unbounded"/>
1665 239     </xs:sequence>
1666 240     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1667 241 </xs:complexType>
1668 242
1669 243 <xs:element name="RelationshipType" type="tRelationshipType"/>
1670 244 <xs:complexType name="tRelationshipType">
1671 245     <xs:complexContent>
1672 246         <xs:extension base="tExtensibleElements">
1673 247             <xs:sequence>
1674 248                 <xs:element name="RelationshipTypeProperties" minOccurs="0">
1675 249                     <xs:complexType>
1676 250                         <xs:attribute name="element" type="xs:QName"/>
1677 251                         <xs:attribute name="type" type="xs:QName"/>
1678 252                     </xs:complexType>
1679 253                 </xs:element>
1680 254                 <xs:element name="InstanceStates"
1681 255                     type="tTopologyElementInstanceStates" minOccurs="0"/>
1682 256             </xs:sequence>
1683 257             <xs:attribute name="id" type="xs:ID" use="required"/>
1684 258             <xs:attribute name="name" type="xs:string" use="optional"/>
1685 259             <xs:attribute name="semantics" type="xs:anyURI" use="required"/>
1686 260             <xs:attribute name="cascadingDeletion" type="tBoolean"
1687 261                 use="optional" default="no"/>
1688 262             <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1689 263         </xs:extension>
1690 264     </xs:complexContent>
1691 265 </xs:complexType>
1692 266
1693 267 <xs:element name="RelationshipTemplate" type="tRelationshipTemplate"/>
1694 268 <xs:complexType name="tRelationshipTemplate">
1695 269     <xs:complexContent>
1696 270         <xs:extension base="tExtensibleElements">
1697 271             <xs:sequence>
1698 272                 <xs:element name="SourceElement">
1699 273                     <xs:complexType>
1700 274                         <xs:attribute name="id" type="xs:IDREF" use="required"/>
1701 275                     </xs:complexType>
1702 276                 </xs:element>
1703 277                 <xs:choice>
1704 278                     <xs:element name="TargetElement">
1705 279                         <xs:complexType>
1706 280                             <xs:attribute name="id" type="xs:IDREF" use="required"/>
1707 281                         </xs:complexType>
1708 282                     </xs:element>
1709 283                     <xs:element name="TargetElementReference">
1710 284                         <xs:complexType>
1711 285                             <xs:attribute name="id" type="xs:QName" use="required"/>

```

```

1712 286         </xs:complexType>
1713 287     </xs:element>
1714 288 </xs:choice>
1715 289 <xs:element name="PropertyDefaults" minOccurs="0">
1716 290     <xs:complexType>
1717 291         <xs:sequence>
1718 292             <xs:any namespace="##other" processContents="lax"/>
1719 293         </xs:sequence>
1720 294     </xs:complexType>
1721 295 </xs:element>
1722 296 <xs:element name="PropertyConstraints" minOccurs="0">
1723 297     <xs:complexType>
1724 298         <xs:sequence>
1725 299             <xs:element name="PropertyConstraint"
1726 300                 type="tPropertyConstraint" maxOccurs="unbounded"/>
1727 301         </xs:sequence>
1728 302     </xs:complexType>
1729 303 </xs:element>
1730 304 <xs:element name="RelationshipConstraints" minOccurs="0">
1731 305     <xs:complexType>
1732 306         <xs:sequence>
1733 307             <xs:element name="RelationshipConstraint"
1734 308                 maxOccurs="unbounded">
1735 309                 <xs:complexType>
1736 310                     <xs:sequence>
1737 311                         <xs:any namespace="##other" processContents="lax"
1738 312                             minOccurs="0"/>
1739 313                     </xs:sequence>
1740 314                     <xs:attribute name="constraintType" type="xs:anyURI"
1741 315                         use="required"/>
1742 316                 </xs:complexType>
1743 317             </xs:element>
1744 318         </xs:sequence>
1745 319     </xs:complexType>
1746 320 </xs:element>
1747 321 </xs:sequence>
1748 322 <xs:attribute name="id" type="xs:ID" use="required"/>
1749 323 <xs:attribute name="name" type="xs:string" use="optional"/>
1750 324 <xs:attribute name="relationshipType" type="xs:QName"
1751 325     use="required"/>
1752 326 </xs:extension>
1753 327 </xs:complexContent>
1754 328 </xs:complexType>
1755 329
1756 330 <xs:element name="NodeTypes" type="tNodeTypes"/>
1757 331 <xs:complexType name="tNodeTypes">
1758 332     <xs:sequence>
1759 333         <xs:element name="NodeType" type="tNodeType" maxOccurs="unbounded"/>
1760 334     </xs:sequence>
1761 335     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1762 336 </xs:complexType>
1763 337
1764 338 <xs:element name="NodeType" type="tNodeType"/>
1765 339 <xs:complexType name="tNodeType">
1766 340     <xs:complexContent>
1767 341         <xs:extension base="tExtensibleElements">
1768 342             <xs:sequence>
1769 343                 <xs:element name="NodeTypeProperties" minOccurs="0">

```

```

1770 344      <xs:complexType>
1771 345      <xs:attribute name="element" type="xs:QName"/>
1772 346      <xs:attribute name="type" type="xs:QName"/>
1773 347      </xs:complexType>
1774 348    </xs:element>
1775 349    <xs:element name="DerivedFrom" minOccurs="0">
1776 350      <xs:complexType>
1777 351        <xs:attribute name="nodeTypeRef" type="xs:QName"
1778 352          use="required"/>
1779 353      </xs:complexType>
1780 354    </xs:element>
1781 355    <xs:element name="InstanceStates"
1782 356      type="tTopologyElementInstanceStates" minOccurs="0"/>
1783 357    <xs:element name="Interfaces" minOccurs="0">
1784 358      <xs:complexType>
1785 359        <xs:sequence>
1786 360          <xs:element name="Interface" maxOccurs="unbounded">
1787 361            <xs:complexType>
1788 362              <xs:choice>
1789 363                <xs:element name="WSDL" type="tWSDL" maxOccurs="unbounded"/>
1790 364                <xs:element name="REST" type="tREST" maxOccurs="unbounded"/>
1791 365                <xs:element name="Operation" maxOccurs="unbounded">
1792 366                  <xs:complexType>
1793 367                    <xs:complexContent>
1794 368                      <xs:extension base="tOperation"/>
1795 369                    </xs:complexContent>
1796 370                  </xs:complexType>
1797 371                </xs:element>
1798 372              </xs:choice>
1799 373            </xs:complexType>
1800 374          </xs:element>
1801 375        </xs:sequence>
1802 376      </xs:complexType>
1803 377    </xs:element>
1804 378    <xs:element name="Policies" minOccurs="0">
1805 379      <xs:complexType>
1806 380        <xs:sequence>
1807 381          <xs:element name="Policy" type="tPolicy"
1808 382            maxOccurs="unbounded"/>
1809 383        </xs:sequence>
1810 384      </xs:complexType>
1811 385    </xs:element>
1812 386    <xs:element name="DeploymentArtifacts" minOccurs="0">
1813 387      <xs:complexType>
1814 388        <xs:sequence>
1815 389          <xs:element name="DeploymentArtifact"
1816 390            type="tDeploymentArtifact" maxOccurs="unbounded"/>
1817 391        </xs:sequence>
1818 392      </xs:complexType>
1819 393    </xs:element>
1820 394  </xs:sequence>
1821 395  <xs:attribute name="id" type="xs:ID" use="required"/>
1822 396  <xs:attribute name="name" type="xs:string" use="optional"/>
1823 397  <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1824 398  </xs:extension>
1825 399  </xs:complexContent>
1826 400 </xs:complexType>
1827 401

```



```

1828 402 <xs:element name="Plans" type="tPlans"/>
1829 403 <xs:complexType name="tPlans">
1830 404   <xs:sequence>
1831 405     <xs:element name="Plan" type="tPlan" maxOccurs="unbounded"/>
1832 406   </xs:sequence>
1833 407   <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1834 408 </xs:complexType>
1835 409
1836 410 <xs:element name="Plan" type="tPlan"/>
1837 411 <xs:complexType name="tPlan">
1838 412   <xs:complexContent>
1839 413     <xs:extension base="tExtensibleElements">
1840 414       <xs:sequence>
1841 415         <xs:element name="Precondition" type="tCondition" minOccurs="0"/>
1842 416         <xs:choice>
1843 417           <xs:element name="PlanModel">
1844 418             <xs:complexType>
1845 419               <xs:sequence>
1846 420                 <xs:any namespace="##other" processContents="lax"/>
1847 421               </xs:sequence>
1848 422             </xs:complexType>
1849 423           </xs:element>
1850 424           <xs:element name="PlanModelReference">
1851 425             <xs:complexType>
1852 426               <xs:attribute name="reference" type="xs:anyURI"
1853 427                 use="required"/>
1854 428             </xs:complexType>
1855 429           </xs:element>
1856 430         </xs:choice>
1857 431       </xs:sequence>
1858 432       <xs:attribute name="id" type="xs:ID" use="required"/>
1859 433       <xs:attribute name="name" type="xs:string" use="optional"/>
1860 434       <xs:attribute name="planType" type="xs:anyURI" use="required"/>
1861 435       <xs:attribute name="languageUsed" type="xs:anyURI" use="required"/>
1862 436     </xs:extension>
1863 437   </xs:complexContent>
1864 438 </xs:complexType>
1865 439
1866 440 <xs:complexType name="tPolicy">
1867 441   <xs:complexContent>
1868 442     <xs:extension base="tExtensibleElements">
1869 443       <xs:attribute name="name" type="xs:string" use="required"/>
1870 444       <xs:attribute name="type" type="xs:anyURI" use="required"/>
1871 445     </xs:extension>
1872 446   </xs:complexContent>
1873 447 </xs:complexType>
1874 448
1875 449 <xs:complexType name="tEnvironmentConstraint">
1876 450   <xs:sequence>
1877 451     <xs:any namespace="##other" processContents="lax"/>
1878 452   </xs:sequence>
1879 453   <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
1880 454 </xs:complexType>
1881 455
1882 456 <xs:complexType name="tExtensions">
1883 457   <xs:complexContent>
1884 458     <xs:extension base="tExtensibleElements">
1885 459       <xs:sequence>

```

```

1886 460     <xs:element name="Extension" type="tExtension"
1887 461         maxOccurs="unbounded"/>
1888 462     </xs:sequence>
1889 463 </xs:extension>
1890 464 </xs:complexContent>
1891 465 </xs:complexType>
1892 466
1893 467 <xs:complexType name="tExtension">
1894 468     <xs:complexContent>
1895 469         <xs:extension base="tExtensibleElements">
1896 470             <xs:attribute name="namespace" type="xs:anyURI" use="required"/>
1897 471             <xs:attribute name="mustUnderstand" type="tBoolean" use="optional"
1898 472                 default="yes"/>
1899 473         </xs:extension>
1900 474     </xs:complexContent>
1901 475 </xs:complexType>
1902 476
1903 477 <xs:complexType name="tParameter">
1904 478     <xs:attribute name="name" type="xs:string" use="required"/>
1905 479     <xs:attribute name="type" type="xs:string" use="required"/>
1906 480     <xs:attribute name="required" type="tBoolean" use="optional"
1907 481         default="yes"/>
1908 482 </xs:complexType>
1909 483
1910 484 <xs:complexType name="tWSDL">
1911 485     <xs:attribute name="portType" type="xs:QName" use="required"/>
1912 486     <xs:attribute name="operation" type="xs:NCName" use="optional"/>
1913 487 </xs:complexType>
1914 488
1915 489 <xs:complexType name="tOperation">
1916 490     <xs:complexContent>
1917 491         <xs:extension base="tExtensibleElements">
1918 492             <xs:sequence>
1919 493                 <xs:element name="Implementations">
1920 494                     <xs:complexType>
1921 495                         <xs:sequence>
1922 496                             <xs:element name="Implementation" maxOccurs="unbounded">
1923 497                                 <xs:complexType>
1924 498                                     <xs:choice>
1925 499                                         <xs:element name="ImplementationProper" type="xs:anyType"
1926 500                                             minOccurs="0"/>
1927 501                                         <xs:element name="ImplementationReference" minOccurs="0">
1928 502                                             <xs:complexType>
1929 503                                                 <xs:attribute name="ref" type="xs:anyURI"/>
1930 504                                             </xs:complexType>
1931 505                                         </xs:element>
1932 506                                     </xs:choice>
1933 507                                         <xs:attribute name="implementationID" type="xs:anyURI"/>
1934 508                                         <xs:attribute name="language" type="xs:anyURI"/>
1935 509                                     </xs:complexType>
1936 510                                         </xs:element>
1937 511                                     </xs:sequence>
1938 512                                 </xs:complexType>
1939 513                             </xs:element>
1940 514                             <xs:element name="InputParameters" minOccurs="0">
1941 515                                 <xs:complexType>
1942 516                                     <xs:sequence>
1943 517                                         <xs:element name="InputParameter" type="tParameter"

```

```

1944 518         maxOccurs="unbounded"/>
1945 519     </xs:sequence>
1946 520 </xs:complexType>
1947 521 </xs:element>
1948 522 <xs:element name="OutputParameters" minOccurs="0">
1949 523     <xs:complexType>
1950 524         <xs:sequence>
1951 525             <xs:element name="OutputParameter" type="tParameter"
1952 526                 maxOccurs="unbounded"/>
1953 527         </xs:sequence>
1954 528     </xs:complexType>
1955 529 </xs:element>
1956 530 </xs:sequence>
1957 531 <xs:attribute name="name" type="xs:NCName" use="required"/>
1958 532 </xs:extension>
1959 533 </xs:complexContent>
1960 534 </xs:complexType>
1961 535
1962 536 <xs:complexType name="tREST">
1963 537     <xs:attribute name="method" default="GET">
1964 538         <xs:simpleType>
1965 539             <xs:restriction base="xs:string">
1966 540                 <xs:enumeration value="GET"/>
1967 541                 <xs:enumeration value="PUT"/>
1968 542                 <xs:enumeration value="POST"/>
1969 543                 <xs:enumeration value="DELETE"/>
1970 544             </xs:restriction>
1971 545         </xs:simpleType>
1972 546     </xs:attribute>
1973 547     <xs:attribute name="requestURI" type="xs:anyURI" use="required"/>
1974 548     <xs:attribute name="requestPayload" type="xs:QName"/>
1975 549     <xs:attribute name="responsePayload" type="xs:QName"/>
1976 550 </xs:complexType>
1977 551
1978 552 <xs:complexType name="tCondition">
1979 553     <xs:sequence>
1980 554         <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1981 555     </xs:sequence>
1982 556     <xs:attribute name="expressionLanguage" type="xs:anyURI"
1983 557         use="required"/>
1984 558 </xs:complexType>
1985 559
1986 560 <xs:complexType name="tTopologyElementInstanceStates">
1987 561     <xs:sequence>
1988 562         <xs:element name="InstanceState" maxOccurs="unbounded">
1989 563             <xs:complexType>
1990 564                 <xs:attribute name="state" type="xs:anyURI" use="required"/>
1991 565             </xs:complexType>
1992 566         </xs:element>
1993 567     </xs:sequence>
1994 568 </xs:complexType>
1995 569
1996 570 <xs:simpleType name="tBoolean">
1997 571     <xs:restriction base="xs:string">
1998 572         <xs:enumeration value="yes"/>
1999 573         <xs:enumeration value="no"/>
2000 574     </xs:restriction>
2001 575 </xs:simpleType>

```

2002	576	
2003	577	<xs:simpleType name="importedURI">
2004	578	<xs:restriction base="xs:anyURI"/>
2005	579	</xs:simpleType>
2006	580	
2007	581	</xs:schema>

---

## Appendix D. Sample

This appendix contains the full sample used in this specification.

### D.1 Sample Service Topology Definition

```
<ServiceTemplate name="myService"
  targetNamespace="http://www.ibm.com/sample">

  <Types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
      <xs:element name="ApplicationProperties">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Owner" type="xs:string"/>
            <xs:element name="InstanceName" type="xs:string"/>
            <xs:element name="AccountID" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="AppServerProperties">
        <xs:complexType>
          <xs:sequence>
            <element name="HostName" type="string"/>
            <element name="IPAddress" type="string"/>
            <element name="HeapSize" type="positiveInteger"/>
            <element name="SoapPort" type="positiveInteger"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </Types>

  <TopologyTemplate id="SampleApplication">

    <NodeTemplate id="MyApplication"
      name="My Application"
      nodeType="abc:Application">
      <PropertyDefaults>
        <ApplicationProperties>
          <Owner>Frank</Owner>
          <InstanceName>Thomas' favorite application</InstanceName>
        </ApplicationProperties>
      </PropertyDefaults>
    </NodeTemplate>

    <NodeTemplate id="MyAppServer"
      name="My Application Server"
      nodeType="abc:ApplicationServer"
      minInstances="0"
      maxInstances="unbounded"/>

    <RelationshipTemplate id="MyDeploymentRelationship"
```

```

2060         relationshipType="deployedOn">
2061     <SourceElement id="MyApplication"/>
2062     <TargetElement id="MyAppServer"/>
2063 </RelationshipTemplate>
2064
2065 </TopologyTemplate>
2066
2067 <NodeTypes>
2068     <NodeType name="Application">
2069         <documentation xml:lang="EN">
2070             A reusable definition of a node type representing an
2071             application that can be deployed on application servers.
2072         </documentation>
2073         <NodeTypeProperties element="ApplicationProperties"/>
2074         <InstanceStates>
2075             <InstanceState state="http://www.my.com/started"/>
2076             <InstanceState state="http://www.my.com/stopped"/>
2077         </InstanceStates>
2078         <Interfaces>
2079             <Interface>
2080                 <Operation name="DeployApplication">
2081                     <InputParameters>
2082                         <InputParamter name="InstanceName"
2083                             type="string"/>
2084                         <InputParamter name="AppServerHostname"
2085                             type="string"/>
2086                         <InputParamter name="ContextRoot"
2087                             type="string"/>
2088                     </InputParameters>
2089                     <Implementations>
2090                         <Implementation>
2091                             ...
2092                         </Implementation>
2093                     </Implementations>
2094                 </Operation>
2095             </Interface>
2096         </Interfaces>
2097     </NodeType>
2098     <NodeType name="ApplicationServer"
2099         targetNamespace="http://www.ibm.com/sample">
2100         <NodeTypeProperties element="AppServerProperties"/>
2101         <Interfaces>
2102             <Interface>
2103                 <Operation name="AcquireNetworkAddress">
2104                     <OutputParameters>
2105                         <OutputParamter name="Hostname"
2106                             type="string"/>
2107                         <OutputParamter name="IPAddress"
2108                             type="string"/>
2109                     </OutputParameters>
2110                     <Implementations>
2111                         <Implementation>
2112                             ...
2113                         </Implementation>
2114                     </Implementations>
2115                 </Operation>
2116                 <Operation name="DeployApplicationServer">
2117                     <InputParameters>

```

```

2118         <InputParamter name="Hostname"
2119             type="string"/>
2120         <InputParamter name="IPAddress"
2121             type="string"/>
2122         <InputParamter name="HeapSize"
2123             type="int"/>
2124         <InputParamter name="SoapPort"
2125             type="int"/>
2126     </InputParameters>
2127     <OutputParameters>
2128         <OutputParamter name="ServerID"
2129             type="string"/>
2130     </OutputParameters>
2131     <Implementations>
2132         <Implementation>
2133             ...
2134         </Implementation>
2135     </Implementations>
2136 </Operation>
2137 </Interface>
2138 </Interfaces>
2139 </NodeType>
2140 </NodeTypes>
2141
2142 <RelationshipTypes>
2143     <documentation xml:lang="EN">
2144         A reusable definition of relation that expresses deployment of
2145         an artifact on a hosting environment.
2146     </documentation>
2147     <RelationshipType name="deployedOn"
2148         semantics="www.my.com/RelSemantics/deployedOn">
2149     </RelationshipType>
2150 </RelationshipTypes>
2151
2152 <Plans>
2153     <Plan id="DeployApplication"
2154         name="Sample Application Build Plan"
2155         planType="http://docs.oasis-
2156             open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
2157         languageUsed="http://www.omg.org/spec/BPMN/2.0/">
2158
2159         <PreCondition expressionLanguage="www.my.com/text">?
2160             Run only if funding is available
2161         </PreCondition>
2162
2163     <PlanModel>
2164         <process name="DeployNewApplication" id="p1">
2165             <documentation>This process deploys a new instance of the
2166             sample application.
2167             </documentation>
2168
2169             <task id="t1" name="CreateAccount"/>
2170
2171             <task id="t2" name="AcquireNetworkAddresses"
2172                 isSequential="false"
2173                 loopDataInput="t2Input.LoopCounter"/>
2174             <documentation>Assumption: t2 gets data of type "input"
2175                 as input and this data has a field names "LoopCounter"

```

```

2176         that contains the actual multiplicity of the task.
2177     </documentation>
2178
2179     <task id="t3" name="DeployApplicationServer"
2180         isSequential="false"
2181         loopDataInput="t3Input.LoopCounter"/>
2182
2183     <task id="t4" name="DeployApplication"
2184         isSequential="false"
2185         loopDataInput="t4Input.LoopCounter"/>
2186
2187     <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
2188     <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
2189     <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
2190 </process>
2191 </PlanModel>
2192 </Plan>
2193
2194 <Plan id="RemoveApplication"
2195     planType="http://docs.oasis-
2196         open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
2197     languageUsed="http://docs.oasis-
2198         open.org/wsbpel/2.0/process/executable">
2199     <PlanModelReference reference="prj:RemoveApp"/>
2200 </Plan>
2201 </Plans>
2202
2203 </ServiceTemplate>

```



---

## Appendix E. Revision History

Revision	Date	Editor	Changes Made
wd-01	2012-01-26	Thomas Spatzier	Changes for JIRA Issue TOSCA-1: Initial working draft based on input spec delivered to TOSCA TC. Copied all content from input spec and just changed namespace. Added line numbers to whole document.
wd-02	2012-02-23	Mike Edwards, Thomas Spatzier	Changes for JIRA Issue TOSCA-6: Reviewed and adapted normative statement keywords according to RFC2119.
wd-03	2012-03-06	Arvind Srinivasan, Mike Edwards, Thomas Spatzier	Changes for JIRA Issue TOSCA-10: Marked all occurrences of keywords from the TOSCA language (element and attribute names) in Courier New font.