

Topology and Orchestration Specification for Cloud Applications Version 1.0

Committee Specification Draft 02

05 April 2012

Specification URIs

This version:

<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd02/TOSCA-v1.0-csd02.doc> (Authoritative)
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd02/TOSCA-v1.0-csd02.html>
<http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd02/TOSCA-v1.0-csd02.pdf>

Previous 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>

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), CA Technologies
Simon Moser (smoser@de.ibm.com), IBM

Editors:

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

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schemas: <http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd02/schemas/>

Declared XML namespaces:

- <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" 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. 05 April 2012. OASIS Committee Specification Draft 02. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd02/TOSCA-v1.0-csd02.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	18
4.3	Derivation Rules	21
4.4	Example	21
5	Relationship Types	23
5.1	Syntax	23
5.2	Properties	23
5.3	Example	24
6	Topology Template	25
6.1	Syntax	25
6.2	Properties	27
6.3	Example	31
7	Plans	33
7.1	Syntax	33
7.2	Properties	33
7.3	Use of Process Modeling Languages	34
7.4	Example	34
8	Security Considerations	36
9	Conformance	37
Appendix A.	Portability and Interoperability Considerations	38
Appendix B.	Acknowledgements	39
Appendix C.	Complete TOSCA Grammar	41
Appendix D.	TOSCA Schema	46

Appendix E.	Sample	58
E.1	Sample Service Topology Definition	58
Appendix F.	Revision History	62

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

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	<code>http://docs.oasis-open.org/tosca/ns/2011/12</code>
xs	<code>http://www.w3.org/2001/XMLSchema</code>
wsdl	<code>http://schemas.xmlsoap.org/wsdl/</code>
bpmn	<code>http://www.omg.org/bpmn/2.0</code>

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>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:element name="ProjectProperties">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Owner" type="xs:string"/>
          <xs:element name="ProjectName" type="xs:string"/>
          <xs:element name="AccountID" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</Types>
```

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

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

Example of use of a documentation:

```
<ServiceTemplate id="myService" name="My Service" ...>
  <documentation xml:lang="EN">
    This is a simple example of the usage of the documentation
    element as nested under a ServiceTemplate element.
  </documentation>
</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.

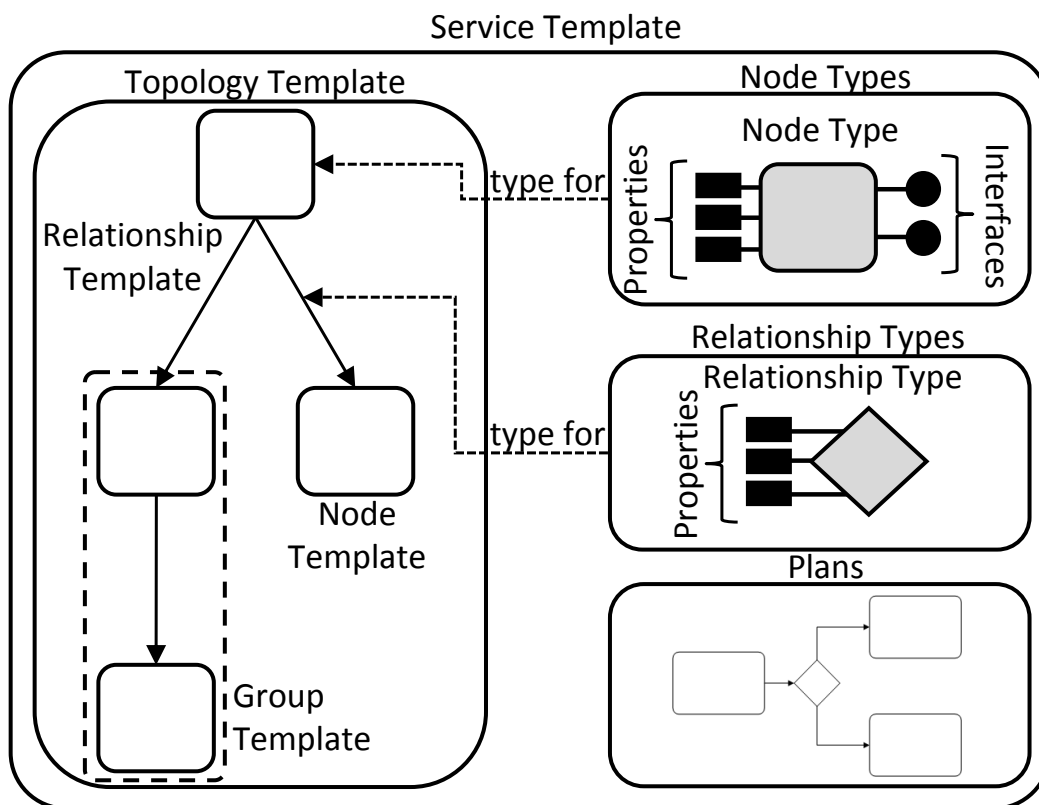


Figure 1: Structural Elements of a Service Template and their Relations

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 elements it connects and the direction of the relationship by defining one source and one target element (in nested `SourceElement` and `TargetElement` 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 name="NCName | anyURI">+
        <Operation name="NCName">+
          (
            <WSDL portType="QName"
              operation="NCName"/>
            |
            <REST method="GET | PUT | POST | DELETE"
              abs_path="anyURI"?
              absoluteURI="anyURI"?
              requestBody="QName"?
              responseBody="QName"?>
              <Parameters>?
                <Parameter name="string" required="yes|no"/>+
              </Parameters>
              <Headers>?
                <Header name="string" required="yes|no"/>+
              </Headers>
            </REST>
            |
            <ScriptOperation>
              <InputParameters>?
```



```

427 44
428 45         <InputParamter name="string"
429 46             type="string"
430 47             required="yes|no"/>+
431 48
432 49     </InputParameters>
433 50
434 51     <OutputParameters>?
435 52
436 53         <OutputParamter name="string"
437 54             type="string"
438 55             required="yes|no"/>+
439 56
440 57     </OutputParameters>
441 58
442 59 </ScriptOperation>
443 60 )
444 61
445 62 </Operation>
446 63
447 64 <ImplementationArtifacts>?
448 65
449 66     <ImplementationArtifact operationName="string"
450 67         type="anyURI">+
451 68
452 69     <RequiredContainerCapabilities>?
453 70         <RequiredContainerCapability capability="anyURI"/>+
454 71     </RequiredContainerCapabilities>
455 72
456 73     artifact specific content
457 74
458 75     <ImplementationArtifact>
459 76
460 77 </ImplementationArtifacts>
461 78
462 79 </Interface>
463 80
464 81 </Interfaces>
465 82
466 83 <Policies>?
467 84     <Policy name="string" type="anyURI">+
468 85         policy specific content
469 86     </Policy>
470 87 </Policies>
471 88
472 89 <DeploymentArtifacts>?
473 90     <DeploymentArtifact name="string" type="anyURI">+
474 91         artifact specific content
475 92     </DeploymentArtifact>
476 93 </DeploymentArtifacts>
477 94
478 95 </NodeType>
479 96
480 97 </NodeTypes>

```

4.2 Properties

The `NodeType` element has the following properties:

- `id`: This attribute specifies the identifier of the Node Type. The identifier of the Node Type MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Node Type.
- `NodeTypeProperties`: These are the observable properties of the Node Type, such as its configuration and state.
- `DerivedFrom`: This is an optional reference to another Node Type from which this Node Type derives. Conflicting definitions are resolved by the rule that local new definitions always override derived definitions. See section 4.3 Derivation Rules for details.
- `InstanceStates`: This optional element lists the set of states an instance of this Node Type can occupy at runtime.
- `Interfaces`: These are the definitions of functions that can be performed on (instances of) this Node Type.
- `Policies`: The nested list of elements provides information related to a particular management aspect like billing or monitoring.
- `DeploymentArtifacts`: This element specifies deployment artifacts relevant for the Node Type. A deployment artifact is an entity that – if specified – is needed for creating an instance of the corresponding Node Type. For example, a virtual image could be a deployment artifact of a JEE server.

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

- The `element` attribute provides the QName of an XML element defining the structure of the Node Type Properties.
- The `type` attribute provides the QName of an XML (complex) type defining the structure of the Node Type Properties.

The `DerivedFrom` element has the following properties:

- `nodeTypeRef`: The QName specifies the Node Type from which this Node Type derives its definitions.

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.

The `Interface` element has the following properties:

- `name`: The name of the interface. This name is either a URI or it is an NCName that MUST be unique in the target namespace.
- `Operation`: This element defines an operation available to manage particular aspects of the Node Type. The `name` attribute of the `Operation` element defines the name of the operation and MUST be unique within the containing `Interface` of the Node Type.
- `ImplementationArtifacts`: This element specifies a set of implementation artifacts for operations in an interface.

521 The `Operation` element has the following properties:

- 522 • `WSDL`: The operation is implemented by means of Web Service technology. The port type and
523 operation of the Web Service are specified in the corresponding attributes of the `WSDL` element.
- 524 • `REST`: The operation is implemented as a REST API.
- 525 • `ScriptOperation`: The operation is implemented by scripts.

526 The `ImplementationArtifacts` element has the following properties:

- 527 • `ImplementationArtifact`: An implementation artifact of an operation. For example, a
528 servlet might be an implementation artifact for a REST API.

529 Multiple implementation artifacts might be required for a single operation, e.g. in case a script
530 operation is realized using different script languages in different environments.
531

532 The `WSDL` element has the following properties:

- 533 • `portType`: This is the QName of the port type that contains the definition of the operation
534 defined as part of the interface. Note that the corresponding namespace MUST be imported.
- 535 • `operation`: This attribute specifies the name of an operation of the port type to become part of
536 the interface.

537 The `REST` element has the following properties:

- 538 • `method`: The HTTP method to be used for building the REST request. If no method is explicitly
539 specified, GET is assumed as default.
- 540 • `abs_path`: The absolute path of the URI that represents the target resource of the request.
541 Note, that the proper network location of the URI MUST be set as value of the Host header field
542 of the request when using `abs_path` instead of `absoluteURI`.
- 543 • `absoluteURI`: The absolute URI of the resource.
544 Note, that either the `abs_path` or the `absoluteURI` MUST be specified.
- 545 • `requestBody`: The data passed in the body of the request message. The QName value of this
546 attribute identifies the specification of the body, e.g. it refers to an XML Schema Definition
547 document.
- 548 • `responseBody`: The data returned in the body of the response message. The QName value of
549 this attribute identifies the specification of the body, e.g. it refers to an XML Schema Definition
550 document.
- 551 • `Parameters`: This nested element describes a list of parameters as nested `Parameter`
552 elements. Each `Parameter` has a `name` attribute and a `required` attribute that indicates
553 whether the parameter is required or not. This list is the base for building the query string of the
554 URI.
- 555 • `Headers`: This nested element describes a list of HTTP request headers as nested `Header`
556 elements. Each `Header` has a `name` attribute and a `required` attribute that indicates whether
557 the header is required or not. Only those headers SHOULD be listed that might be important for
558 specifying the semantics of the request; otherwise, the HTTP client will set HTTP headers as
559 usual.

560 The `ScriptOperation` element has the following properties:

- 561 • `InputParameters`: This optional property contains one or more nested `InputParameter`
562 elements. Each such element specifies three attributes: the `name` of the parameter, its `type`,
563 and whether it must be available as input (`required` attribute with a value of “yes”, which is the
564 default) or not (value “no”). Note that the types of the parameters specified for an operation
565 MUST comply with the type systems of the languages of implementations.
- 566 • `OutputParameters`: This optional property contains one or more nested
567 `OutputParameter` elements. Each such element specifies three attributes: the `name` of the
568 parameter, its `type`, and whether it must be available as output (`required` attribute with a
569 value of “yes”, which is the default) or not (value “no”). Note that the types of the parameters
570 specified for an operation MUST comply with the type systems of the languages of
571 implementations.

572 The `ImplementationArtifact` element has the following properties:

- 573 • `operationName`: The name of the operation that is implemented by the actual implementation
574 artifact.
- 575 • `type`: The type of the implementation artifact determines the specific content of the
576 `ImplementationArtifact` element. For example, a script might be provided in place or by
577 reference. A corresponding value of the `type` attribute indicates this.
- 578 • `RequiredContainerCapabilities`: An implementation of an operation might depend on
579 certain capabilities of the environment it is executed in. For example, an implementation of an
580 operation might use a particular interface for manipulating images, EJBs etc.
581 Each such dependency is explicitly declared by a separate
582 `RequiredContainerCapability` element. The `capability` attribute of this element is
583 a URI that denotes the corresponding requirement on the environment.

584 The `Policy` element has the following properties:

- 585 • The `type` attribute specifies the kind of policy (e.g. management practice) supported by an
586 instance of the Node Type containing this element. The `name` attribute defines the name of the
587 policy. The name value MUST be unique within a given Node Type containing the current
588 definition of the Policy.
589 Consider a hypothetical billing policy. In this example the type `www.sample.com/BillingPractice`
590 could define a policy for billing usage of a service instance. The policy specific content can define
591 the interface providing the operations to perform billing. Further content could specify the
592 granularity of the base for payment, e.g. it could provide an enumeration with the possible values
593 “service”, “resource”, and “labor”. A value of “service” might specify that an instance of the
594 corresponding node will be billed during its instance lifetime. A value of “resource” might specify
595 that the resources consumed by an instance will be billed. A value of “labor” might specify that the
596 use of a plan affecting a node instance will be billed.

597 The `DeploymentArtifact` element has the following properties:

- 598 • `name`: The attribute specifies the name of the artifact. Note, that uniqueness of the name within
599 the scope of the encompassing Node Type SHOULD be guaranteed by the definition.
- 600 • `type`: The attribute specifies the type of the deployment artifact definition that is related to the
601 Node Type, i.e. the attribute gives a hint how to interpret the body of the
602 `DeploymentArtifact` element.
603 Note, that the combination of name and type SHOULD be unique within the scope of the Node
604 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 the Node Type under definition consists of the set union of the instances states defined by the Nodes Type derived from and the instance states defined by the Node Type under definition. A set of instance states of the same name will be combined into a single instance state of the same name.
- **Interfaces:** The set of interfaces of the Node Type under definition consists of the set union of interfaces defined by the Node Type derived from and the interfaces defined by the Node Type under definition.
Two interfaces of the same name will be combined into a single, derived interface with the same name. The set of operations of the derived interface consists of the set union of operations defined by both interfaces. If an operation defined by the Node Type under definition has the same name as an operation of the Node Type derived from, the former operation substitutes the latter one.
- **Implementation Artifacts:** The set of implementation artifacts of the Node Type under definition consists of the set union of implementation artifacts defined by the Node Type derived from and the implementation artifacts defined by the Node Type under definition.
If an implementation artifact defined by the Node Type under definition has the same operation name and type as an implementation artifact of the Node Type derived from, the former implementation artifact substitutes the latter one.
- **Deployment Artifacts:** The set of deployment artifacts of the Node Type under definition consists of the set union of the deployment artifacts defined by the Nodes Type derived from and the deployment artifacts defined by the Node Type under definition. A deployment artifact defined by the Node Type under definition substitutes a deployment artifact with the same name and type of the Node Type derived from.
- **Policies:** The set of policies of the Node Type under definition consists of the set union of the policies defined by the Nodes Type derived from and the policies defined by the Node Type under definition. A policy defined by the Node Type under definition 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"
```

```

649         targetNamespace="http://www.ibm.com/sample">
650
651     <NodeTypes>
652
653         <NodeType id="Project" name="My Project">
654
655             <documentation xml:lang="EN">
656                 A reusable definition of a node type supporting
657                 the creation of new projects.
658             </documentation>
659
660             <NodeTypeProperties element="ProjectProperties"/>
661
662             <InstanceStates>
663                 <InstanceState state="www.my.com/active"/>
664                 <InstanceState state="www.my.com/onHalt"/>
665             </InstanceStates>
666
667             <Interfaces>
668                 <Interface name="ProjectInterface">
669                     <Operation name="CreateProject">
670                         <ScriptOperation>
671                             <InputParameters>
672                                 <InputParamter name="ProjectName"
673                                     type="string"/>
674                                 <InputParamter name="Owner"
675                                     type="string"/>
676                                 <InputParamter name="AccountID"
677                                     type="string"/>
678                             </InputParameters>
679                         </ScriptOperation>
680                     </Operation>
681                     <ImplementationArtifacts>
682                         <ImplementationArtifact operationName="CreateProject"
683                             type="http://www.my.com/ScriptArtifact/PhythonReference">
684                             scripts/phython/createProject.py
685                         </ImplementationArtifact>
686                     </ImplementationArtifacts>
687                 </Interface>
688             </Interfaces>
689
690         </NodeType>
691
692     </NodeTypes>
693
694 </ServiceTemplate>

```

695 The Node Type "Project" has three Node Type Properties defined as an XML element in the Types
696 element definition of the Service Template document: Owner, ProjectName and AccountID which are all
697 of type "string". An instance of the Node Type "Project" could be "active" (more precise in state
698 www.my.com/active) or "on hold" (more precise in state "www.my.com/onHold"). A single Interface is
699 defined for this Node Type, and this Interface is defined by an Operation, i.e. its actual implementation is
700 defined by the definition of the Operation. The Operation has the name CreateProject and two Input
701 Parameters (exploiting the default value "yes" of the attribute required of the InputParameter
702 element). The names of these two Input Parameters are ProjectName and AccountID, both of type
703 "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      <ImplementationArtifacts>?
43          <ImplementationArtifact operationName="string"
44              type="anyURI">+
```

```

822 45         <RequiredContainerCapabilities>?
823 46         <RequiredContainerCapability capability="anyURI"/>+
824 47         </RequiredContainerCapabilities>
825 48         artifact specific content
826 49         <ImplementationArtifact>
827 50         </ImplementationArtifacts>
828 51
829 52     </NodeTemplate>
830 53 |
831 54     <RelationshipTemplate id="ID"
832 55         name="string"?
833 56         relationshipType="QName">
834 57
835 58         <SourceElement id="IDREF"/>
836 59
837 60         ( <TargetElement id="IDREF"/>
838 61         |
839 62         <TargetElementReference id="QName"/>
840 63         )
841 64
842 65         <PropertyDefaults>?
843 66         XML fragment
844 67         </PropertyDefaults>
845 68
846 69         <PropertyConstraints>?
847 70
848 71         <PropertyConstraint property="string"
849 72             constraintType="anyURI">+
850 73             constraint?
851 74         </PropertyConstraint>
852 75
853 76         </PropertyConstraints>
854 77
855 78         <RelationshipConstraints>?
856 79
857 80         <RelationshipConstraint constraintType="anyURI">+
858 81             constraint?
859 82         </RelationshipConstraint>
860 83
861 84         </RelationshipConstraints>
862 85
863 86     </RelationshipTemplate>
864 87 |
865 88     <GroupTemplate id="ID"
866 89         name="string"?
867 90         minInstances="int"?
868 91         maxInstances="int|string"?>
869 92
870 93         (
871 94         <NodeTemplate ... />
872 95         |
873 96         <RelationshipTemplate ... />
874 97         |
875 98         <GroupTemplate ... />
876 99         )+
877 100
878 101     <Policies>?

```

```

879 102         <Policy name="string" type="anyURI">+
880 103             policy specific content
881 104         </Policy>
882 105     </Policies>
883 106
884 107     </GroupTemplate>
885 108 )+
886 109
887 110 </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.

- **ImplementationArtifact:** An implementation of an operation. For example, a servlet might be an implementation artifact for a REST API. Multiple implementation artifacts might be

required for a single operation, e.g. in case a script operation is realized using different script languages in different environments.

The `operationName` attribute specifies the name of the operation that is implemented by the implementation artifact under definition. The `type` attribute determines the specific content of the `ImplementationArtifact` element. For example, a script might be provided in place or by reference. A corresponding value of the `type` attribute indicates this.

The nested `RequiredContainerCapabilities` element specifies certain capabilities of the environment an implementation of an operation might depend on. For example, an implementation of an operation might use a particular interface for manipulating images, EJBs etc. Each such dependency is explicitly declared by a separate `RequiredContainerCapability` element. The `capability` attribute of this element is a URI that denotes the corresponding requirement on the environment.

Note, that an implementation artifact specified with the Node Template under definition overrides any implementation artifact with the same `operationName` and the same `type` specified with the Node Type given as value of the `nodeType` attribute of the Node Template under definition.

Otherwise, the implementation artifacts of the Node Type given as value of the `nodeType` attribute of the Node Template under definition and the implementation 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 certain scope. The constraint type specific content of the respective `PropertyConstraint` element could then define the actual scope in which uniqueness has to be ensured in more detail.

The `Policy` element has the following properties:

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

The `EnvironmentConstraint` element has the following properties:

- `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 constraint content.

The `RelationshipTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Relationship Template. The identifier of the Relationship Template MUST be unique within the target namespace.
- `name`: This optional attribute specifies the name of the Relationship Template.

- 1017 • `relationshipType`: The QName value of this property refers to the Relationship Type
1018 providing the type of the Relationship Template.
 - 1019 • `SourceElement`: The `id` attribute of this element references a Node Template or Group
1020 Template within the same Service Template document that is the source of the Relationship
1021 Template.
 - 1022 • `TargetElement`: The `id` attribute of this element references a Node Template or Group
1023 Template within the same Service Template document that is the target of the Relationship
1024 Template.
 - 1025 • `TargetElementReference`: The `id` attribute of this element refers by QName to an
1026 imported Node Template or Group Template that is the target of the Relationship Template. The
1027 referenced Node Template or Group Template will typically be the root node or root group of the
1028 corresponding Topology Template. In some cases a non-root Node Template or non-root Group
1029 Template might be referenced to support access to particular resources from a larger service, for
1030 example. Either `TargetElement` or `TargetElementReference` MUST be specified but
1031 not both.
 - 1032 • `PropertyDefaults`: Specifies initial values for one or more of the Relationship Type
1033 properties of the Relationship Type providing the property definitions in the concrete context of
1034 the Relationship Template.

1035 The initial values are specified by providing an instance document of the XML schema of the
1036 corresponding Relationship Type properties.

1037 The instance document of the XML schema might not validate against the existence constraints
1038 of the corresponding schema: not all relationship type properties might have an initial value
1039 assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the
1040 Property Defaults element. Once the defined Relationship Template has been instantiated, any
1041 XML representation of the Relationship Type properties MUST validate according to the
1042 associated XML schema definition.
 - 1043 • `PropertyConstraints`: Specifies constraints on the use of one or more of the Relationship
1044 Type properties of the Relationship Type providing the property definitions for the Relationship
1045 Template.

1046 Each constraint is specified by means of a separate nested `PropertyConstraint` element.
1047 This element contains the actual encoding of the constraint.
 - 1048 • `RelationshipConstraints`: Specifies constraints on the use of the relationship.

1049 Each constraint is specified by means of a separate nested `RelationshipConstraint`
1050 element. This element can contain the actual encoding of the constraint, or its
1051 `constraintType` attribute already denotes the constraint itself. The constraint type is
1052 specified by means of a URI, which defines both the semantic meaning of the constraint as well
1053 as the format of any content.
- 1054 The `GroupTemplate` element has the following properties:
- 1055 • `id`: This attribute specifies the identifier of the Group Template. The identifier of the Group
1056 Template MUST be unique within the target namespace.
 - 1057 • `name`: This optional attribute specifies the name of the Group Template.
 - 1058 • `minInstances`: This integer attribute specifies the minimum number of instances to be created
1059 when instantiating the Group Template. The default value of this attribute is 1. The value of
1060 `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">

    <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">
```

```
1111         minInstances="0"
1112         maxInstances="unbounded"/>
1113
1114     <RelationshipTemplate id="MyDeploymentRelationship"
1115                         relationshipType="deployedOn">
1116         <SourceElement id="MyApplication"/>
1117         <TargetElement id="MyAppServer"/>
1118     </RelationshipTemplate>
1119
1120 </TopologyTemplate>
1121
1122 </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.

1167 ○ <http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan> - This URI defines the
1168 *build plan* plan type for plans used to initially create a new instance of a service from a
1169 Service Template.

1170 ○ <http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan> - This URI
1171 defines the *termination plan* plan type for plans used to terminate the existence of a
1172 service instance.

1173 Note that all other plan types for managing service instances throughout their life time will be
1174 considered and referred to as *modification plans* in general.

1175 • **languageUsed**: This attribute denotes the process modeling language (or metamodel) used to
1176 specify the plan. For example, "<http://www.omg.org/spec/BPMN/2.0/>" would specify that BPMN
1177 2.0 has been used to model the plan.

1178 • **PreCondition**: This optional element specifies a condition that needs to be satisfied in order
1179 for the plan to be executed. The **expressionLanguage** attribute of this element specifies the
1180 expression language the nested condition is provided in.

1181 Typically, the precondition will be an expression in the instance state attribute of some of the
1182 node templates or relationship templates of the topology template. It will be evaluated based on
1183 the actual values of the corresponding attributes at the time the plan is requested to be executed.
1184 Note, that any other kind of pre-condition is allowed.

1185 • **PlanModel**: This property contains the actual model content.

1186 • **PlanModelReference**: This property points to the model content. Its reference attribute
1187 contains a URI of the model of the plan.
1188

1189 An instance of the **Plan** element MUST either contain the actual plan as instance of the
1190 **PlanModel** element, or point to the model via the **PlanModelReference** element.

1191 7.3 Use of Process Modeling Languages

1192 TOSCA does not specify a separate metamodel for defining plans. Instead, it is assumed that a process
1193 modelling language (a.k.a. metamodel) like BPEL [BPEL 2.0] or BPMN [BPMN 2.0] is used to define
1194 plans. The specification favours the use of BPMN for modeling plans.

1195 7.4 Example

1196 The following defines two Plans, one Plan for creating a new instance of the "SampleApplication"
1197 Topology Template (the plan is named "DeployApplication"), and one Plan for removing instances of
1198 "SampleApplication". The Plan "DeployApplication" is a build plan specified in BPMN; the process model
1199 is immediately included in the Plan Model (note that the BPMN model is incomplete but used to show the
1200 mechanism of the **PlanModel** element). The Plan can only run when the PreCondition "Run only if
1201 funding is available" is satisfied. The Plan "RemoveApplication" is a termination plan specified in BPEL;
1202 the corresponding BPEL definition is defined elsewhere and only referenced by the
1203 **PlanModelReference** element.

```
1204 <Plans>
1205
1206   <Plan id="DeployApplication"
1207     name="Sample Application Build Plan"
1208     planType=
1209       "http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
1210     languageUsed="http://www.omg.org/spec/BPMN/2.0/">
1211
1212     <PreCondition expressionLanguage="www.my.com/text">?
```

```

1213     Run only if funding is available
1214 </PreCondition>
1215
1216 <PlanModel>
1217   <process name="DeployNewApplication" id="p1">
1218     <documentation>This process deploys a new instance of the
1219       sample application.
1220     </documentation>
1221
1222     <task id="t1" name="CreateAccount"/>
1223
1224     <task id="t2" name="AcquireNetworkAddresses"
1225       isSequential="false"
1226       loopDataInput="t2Input.LoopCounter"/>
1227     <documentation>Assumption: t2 gets data of type "input"
1228       as input and this data has a field names "LoopCounter"
1229       that contains the actual multiplicity of the task.
1230     </documentation>
1231
1232     <task id="t3" name="DeployApplicationServer"
1233       isSequential="false"
1234       loopDataInput="t3Input.LoopCounter"/>
1235
1236     <task id="t4" name="DeployApplication"
1237       isSequential="false"
1238       loopDataInput="t4Input.LoopCounter"/>
1239
1240     <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
1241     <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
1242     <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
1243   </process>
1244 </PlanModel>
1245 </Plan>
1246
1247 <Plan id="RemoveApplication"
1248   planType="http://docs.oasis-
1249     open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
1250   languageUsed=
1251     "http://docs.oasis-open.org/wsbpel/2.0/process/executable">
1252   <PlanModelReference reference="prj:RemoveApp"/>
1253 </Plan>
1254
1255 </Plans>

```

1256

8 Security Considerations

1257

TOSCA does not mandate the use of any specific mechanism or technology for client authentication.

1258

However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the infrastructure.

1259

9 Conformance

1260

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. Acknowledgements

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

Participants:

Adolf Hohl	NetApp
Afkham Azeez	WSO2
Allen Bannon	SAP AG
Anthony Rutkowski	Yaana Technologies, LLC
Arvind Srinivasan	IBM
Celso Rodriguez	ASG Software Solutions
Chandrasekhar Sundaresh	CA Technologies
Charith Wickramarachchi	WSO2
Colin Hopkinson	3M HIS
Dale Moberg	Axway Software
Dee Schur	OASIS
Denis Weerasiri	WSO2
Derek Palma	Vnomic
Dhiraj Pathak	PricewaterhouseCoopers LLP:
Diane Mueller	ActiveState Software, Inc.
Doug Davis	IBM
Efraim Moscovich	CA Technologies
Frank Leymann	IBM
Gerd Breiter	IBM
James Thomason	Gale Technologies
Jim Marino	Individual
Joseph Malek	VCE
Kevin Poulter	SAP AG
Koert Struijk	CA Technologies
Lee Thompson	Morphlabs, Inc.
Marvin Waschke	CA Technologies
Mascot Yu	Huawei Technologies Co., Ltd.
Matthew Dovey	JISC Executive, University of Bristol
Matthew Rutkowski	IBM
Michael Schuster	SAP AG
Mike Edwards	IBM
Naveen Joy	Cisco Systems
Nikki Heron	rPath, Inc.
Pascal Vitoux	ASG Software Solutions
Paul Fremantle	WSO2
Paul Lipton	CA Technologies
Rachid Sijelmassi	CA Technologies
Ravi	Cisco Systems
Richard Bill	Jericho Systems
Richard Probst	SAP AG
Roland Wartenberg	Citrix Systems
Satoshi Konno	Morphlabs, Inc.
Sean Shen	China Internet Network Information Center(CNNIC)
Selvaratnam Uthaiyashankar	WSO2

Senaka Fernando	WSO2
Sherry Yu	Red Hat
Simon Moser	IBM
Srinath Perera	WSO2
Stephen Tyler	CA Technologies
Steve Fanshier	Software AG, Inc.
Steve Jones	Capgemini
Steve Winkler	SAP AG
Ted Streete	VCE
Thilina Buddhika	WSO2
Thomas Spatzier	IBM
Tobias Kunze	Red Hat
Wang Xuan	Primeton Technologies, Inc.
Wayne Adams	EMC
Wenbo Zhu	Google Inc.
Xiaonan Song	Primeton Technologies, Inc.
YanJiong Wang	Primeton Technologies, Inc.
Zhexuan Song	Huawei Technologies Co., Ltd.

1274

1275

Appendix C. Complete TOSCA Grammar

```
1277 1 <ServiceTemplate id="ID"
1278 2     name="string"?
1279 3     targetNamespace="anyURI">
1280 4
1281 5     <Extensions>?
1282 6         <Extension namespace="anyURI"
1283 7             mustUnderstand="yes|no"?/>+
1284 8     </Extensions>
1285 9
1286 10    <Import namespace="anyURI"?
1287 11        location="anyURI"?
1288 12        importType="anyURI"/>*
1289 13
1290 14    <Types>?
1291 15        <xs:schema .../>*
1292 16    </Types>
1293 17
1294 18    (
1295 19        <TopologyTemplateReference reference="QName"/>
1296 20        |
1297 21        <TopologyTemplate id="ID"
1298 22            name="string"?>
1299 23
1300 24            (
1301 25                <NodeTemplate id="ID"
1302 26                    name="string"?
1303 27                    nodeType="QName"
1304 28                    minInstances="int"?
1305 29                    maxInstances="int|string"?>
1306 30
1307 31                    <PropertyDefaults>?
1308 32                        XML fragment
1309 33                    </PropertyDefaults>
1310 34
1311 35                    <PropertyConstraints>?
1312 36
1313 37                        <PropertyConstraint property="string"
1314 38                            constraintType="anyURI">+
1315 39                            constraint?
1316 40                        </PropertyConstraint>
1317 41
1318 42                    </PropertyConstraints>
1319 43
1320 44                    <Policies>?
1321 45                        <Policy name="string" type="anyURI">+
1322 46                            policy specific content
1323 47                        </Policy>
1324 48                    </Policies>
1325 49
1326 50                    <EnvironmentConstraints>?
1327 51                        <EnvironmentConstraint constraintType="anyURI">+
1328 52                            constraint type specific content?
1329 53                    </EnvironmentConstraint>
```

```

1330 54      </EnvironmentConstraints>
1331 55
1332 56      <DeploymentArtifacts>?
1333 57          <DeploymentArtifact name="string" type="anyURI">+
1334 58              artifact specific content
1335 59          </DeploymentArtifact>
1336 60      </DeploymentArtifacts>
1337 61
1338 62      <ImplementationArtifacts>?
1339 63          <ImplementationArtifact operationName="string"
1340 64              type="anyURI">+
1341 65              <RequiredContainerCapabilities>?
1342 66                  <RequiredContainerCapability capability="anyURI"/>+
1343 67              </RequiredContainerCapabilities>
1344 68              artifact specific content
1345 69          <ImplementationArtifact>
1346 70      </ImplementationArtifacts>
1347 71
1348 72  </NodeTemplate>
1349 73  |
1350 74  <RelationshipTemplate id="ID"
1351 75      name="string"?
1352 76      relationshipType="QName">+
1353 77
1354 78      <SourceElement id="IDREF"/>
1355 79
1356 80      ( <TargetElement id="IDREF"/>
1357 81          |
1358 82          <TargetElementReference id="QName"/>
1359 83      )
1360 84
1361 85      <PropertyDefaults>?
1362 86          XML fragment
1363 87      </PropertyDefaults>
1364 88
1365 89      <PropertyConstraints>?
1366 90
1367 91          <PropertyConstraint property="string"
1368 92              constraintType="anyURI">+
1369 93              constraint?
1370 94          </PropertyConstraint>
1371 95
1372 96      </PropertyConstraints>
1373 97
1374 98      <RelationshipConstraints>?
1375 99
1376 100          <RelationshipConstraint constraintType="anyURI">+
1377 101              constraint?
1378 102          </RelationshipConstraint>
1379 103
1380 104      </RelationshipConstraints>
1381 105
1382 106  </RelationshipTemplate>
1383 107  |
1384 108  <GroupTemplate id="ID"
1385 109      name="string"?
1386 110      minInstances="int"?

```

```

1387 111             maxInstances="int|string"?>
1388 112
1389 113         (
1390 114             <NodeTemplate ... />
1391 115         |
1392 116             <RelationshipTemplate ... />
1393 117         |
1394 118             <GroupTemplate ... />
1395 119         )+
1396 120
1397 121         <Policies>?
1398 122             <Policy name="string" type="anyURI">+
1399 123                 policy specific content
1400 124             </Policy>
1401 125         </Policies>
1402 126
1403 127     </GroupTemplate>
1404 128 )+
1405 129
1406 130 </TopologyTemplate>
1407 131 )?
1408 132
1409 133 <NodeTypes>?
1410 134
1411 135     <NodeType id="ID"
1412 136         name="string"?>+
1413 137
1414 138         <NodeTypeProperties element="QName"?
1415 139             type="QName"?/>?
1416 140
1417 141         <DerivedFrom nodeTypeRef="QName"/>?
1418 142
1419 143         <InstanceStates>?
1420 144             <InstanceState state="anyURI">+
1421 145         </InstanceStates>
1422 146
1423 147         <Interfaces>?
1424 148
1425 149             <Interface name="NCName | anyURI">+
1426 150
1427 151                 <Operation name="NCName">+
1428 152
1429 153                     (
1430 154                         <WSDL portType="QName"
1431 155                             operation="NCName"/>
1432 156                     |
1433 157                         <REST method="GET | PUT | POST | DELETE"
1434 158                             abs_path="anyURI"?
1435 159                             absoluteURI="anyURI"?
1436 160                             requestBody="QName"?
1437 161                             responseBody="QName"?>
1438 162
1439 163                         <Parameters>?
1440 164                             <Parameter name="string" required="yes|no" />+
1441 165                         </Parameters>
1442 166
1443 167                     <Headers>?

```

```

1444 168         <Header name="string" required="yes|no"/>+
1445 169         </Headers>
1446 170
1447 171     </REST>
1448 172     |
1449 173     <ScriptOperation>
1450 174
1451 175         <InputParameters>?
1452 176
1453 177             <InputParamter name="string"
1454 178                 type="string"
1455 179                 required="yes|no"/>+
1456 180
1457 181         </InputParameters>
1458 182
1459 183         <OutputParameters>?
1460 184
1461 185             <OutputParamter name="string"
1462 186                 type="string"
1463 187                 required="yes|no"/>+
1464 188
1465 189         </OutputParameters>
1466 190
1467 191     </ScriptOperation>
1468 192 )
1469 193
1470 194 </Operation>
1471 195
1472 196 <ImplementationArtifacts>?
1473 197
1474 198     <ImplementationArtifact operationName="string"
1475 199         type="anyURI">+
1476 200
1477 201     <RequiredContainerCapabilities>?
1478 202         <RequiredContainerCapability capability="anyURI"/>+
1479 203     </RequiredContainerCapabilities>
1480 204
1481 205         artifact specific content
1482 206
1483 207     <ImplementationArtifact>
1484 208
1485 209     </ImplementationArtifacts>
1486 210
1487 211 </Interface>
1488 212
1489 213 </Interfaces>
1490 214
1491 215 <DeploymentArtifacts>?
1492 216     <DeploymentArtifact name="string" type="anyURI">+
1493 217         artifact specific content
1494 218     </DeploymentArtifact>
1495 219 </DeploymentArtifacts>
1496 220
1497 221
1498 222 <Policies>?
1499 223
1500 224     <Policy name="string" type="anyURI">+

```

```

1501 225         policy specific content
1502 226     </Policy>
1503 227
1504 228     </Policies>
1505 229
1506 230 </NodeType>
1507 231
1508 232 </NodeTypes>
1509 233
1510 234 <RelationshipTypes>?
1511 235
1512 236     <RelationshipType id="ID"
1513 237                         name="string"?
1514 238                         semantics="anyURI"
1515 239                         cascadingDeletion="yes|no"?>+
1516 240
1517 241         <RelationshipTypeProperties element="QName"?
1518 242                                     type="QName"?/>?
1519 243
1520 244         <InstanceStates>?
1521 245             <InstanceState state="anyURI">+
1522 246         </InstanceStates>
1523 247
1524 248     </RelationshipType>
1525 249
1526 250 </RelationshipTypes>
1527 251
1528 252 <Plans>?
1529 253
1530 254     <Plan id="ID"
1531 255             name="string"?
1532 256             planType="anyURI"
1533 257             languageUsed="anyURI">+
1534 258
1535 259         <PreCondition expressionLanguage="anyURI">?
1536 260             condition
1537 261         </PreCondition>
1538 262
1539 263         ( <PlanModel>
1540 264             actual plan
1541 265         </PlanModel>
1542 266             |
1543 267             <PlanModelReference reference="anyURI"/>
1544 268         )
1545 269
1546 270     </Plan>
1547 271
1548 272 </Plans>
1549 273
1550 274 </ServiceTemplate>

```

Appendix D. TOSCA Schema

1551

```
1552 1 <?xml version="1.0" encoding="UTF-8"?>
1553 2 <xs:schema targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
1554 3   elementFormDefault="qualified" attributeFormDefault="unqualified"
1555 4   xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
1556 5   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1557 6
1558 7   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
1559 8     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
1560 9
1561 10  <xs:element name="documentation" type="tDocumentation"/>
1562 11  <xs:complexType name="tDocumentation" mixed="true">
1563 12    <xs:sequence>
1564 13      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1565 14    </xs:sequence>
1566 15    <xs:attribute name="source" type="xs:anyURI"/>
1567 16    <xs:attribute ref="xml:lang"/>
1568 17  </xs:complexType>
1569 18
1570 19  <xs:complexType name="tExtensibleElements">
1571 20    <xs:sequence>
1572 21      <xs:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
1573 22      <xs:any namespace="##other" processContents="lax" minOccurs="0"
1574 23        maxOccurs="unbounded"/>
1575 24    </xs:sequence>
1576 25    <xs:anyAttribute namespace="##other" processContents="lax"/>
1577 26  </xs:complexType>
1578 27
1579 28  <xs:complexType name="tImport">
1580 29    <xs:complexContent>
1581 30      <xs:extension base="tExtensibleElements">
1582 31        <xs:attribute name="namespace" type="xs:anyURI"/>
1583 32        <xs:attribute name="location" type="xs:anyURI"/>
1584 33        <xs:attribute name="importType" type="importedURI" use="required"/>
1585 34      </xs:extension>
1586 35    </xs:complexContent>
1587 36  </xs:complexType>
1588 37
1589 38  <xs:element name="ServiceTemplate">
1590 39    <xs:complexType>
1591 40      <xs:complexContent>
1592 41        <xs:extension base="tServiceTemplate"/>
1593 42      </xs:complexContent>
1594 43    </xs:complexType>
1595 44  </xs:element>
1596 45
1597 46  <xs:complexType name="tServiceTemplate">
1598 47    <xs:complexContent>
1599 48      <xs:extension base="tExtensibleElements">
1600 49        <xs:sequence>
1601 50          <xs:element name="Import" type="tImport" minOccurs="0"
1602 51            maxOccurs="unbounded"/>
1603 52          <xs:element name="Types" minOccurs="0">
```

```

1604 53      <xs:complexType>
1605 54      <xs:sequence>
1606 55          <xs:any namespace="##other" processContents="lax" minOccurs="0"
1607 56              maxOccurs="unbounded"/>
1608 57      </xs:sequence>
1609 58      </xs:complexType>
1610 59      </xs:element>
1611 60      <xs:element name="Extensions" minOccurs="0">
1612 61          <xs:complexType>
1613 62              <xs:sequence>
1614 63                  <xs:element name="Extension" type="tExtension"
1615 64                      maxOccurs="unbounded"/>
1616 65              </xs:sequence>
1617 66          </xs:complexType>
1618 67      </xs:element>
1619 68      <xs:choice minOccurs="0">
1620 69          <xs:element name="TopologyTemplateReference">
1621 70              <xs:complexType>
1622 71                  <xs:attribute name="reference" type="xs:QName"/>
1623 72              </xs:complexType>
1624 73          </xs:element>
1625 74          <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1626 75      </xs:choice>
1627 76      <xs:element name="NodeTypes" type="tNodeTypes" minOccurs="0"/>
1628 77      <xs:element name="RelationshipTypes" type="tRelationshipTypes"
1629 78          minOccurs="0"/>
1630 79      <xs:element name="Plans" type="tPlans" minOccurs="0"/>
1631 80      </xs:sequence>
1632 81      <xs:attribute name="id" type="xs:ID" use="required"/>
1633 82      <xs:attribute name="name" type="xs:string" use="optional"/>
1634 83      <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1635 84      </xs:extension>
1636 85      </xs:complexContent>
1637 86      </xs:complexType>
1638 87
1639 88      <xs:complexType name="tDeploymentArtifact">
1640 89          <xs:complexContent>
1641 90              <xs:extension base="tExtensibleElements">
1642 91                  <xs:attribute name="name" type="xs:string" use="required"/>
1643 92                  <xs:attribute name="type" type="xs:anyURI" use="required"/>
1644 93              </xs:extension>
1645 94          </xs:complexContent>
1646 95      </xs:complexType>
1647 96
1648 97      <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1649 98      <xs:complexType name="tNodeTemplate">
1650 99          <xs:complexContent>
1651 100              <xs:extension base="tExtensibleElements">
1652 101                  <xs:sequence>
1653 102                      <xs:element name="PropertyDefaults" minOccurs="0">
1654 103                          <xs:complexType>
1655 104                              <xs:sequence>
1656 105                                  <xs:any namespace="##other" processContents="lax"/>
1657 106                              </xs:sequence>
1658 107                          </xs:complexType>
1659 108                      </xs:element>
1660 109                      <xs:element name="PropertyConstraints" minOccurs="0">

```

```

1661 110      <xs:complexType>
1662 111      <xs:sequence>
1663 112          <xs:element name="PropertyConstraint"
1664 113              type="tPropertyConstraint" maxOccurs="unbounded"/>
1665 114      </xs:sequence>
1666 115  </xs:complexType>
1667 116 </xs:element>
1668 117 <xs:element name="Policies" minOccurs="0">
1669 118     <xs:complexType>
1670 119         <xs:sequence>
1671 120             <xs:element name="Policy" type="tPolicy"
1672 121                 maxOccurs="unbounded"/>
1673 122         </xs:sequence>
1674 123     </xs:complexType>
1675 124 </xs:element>
1676 125 <xs:element name="EnvironmentConstraints" minOccurs="0">
1677 126     <xs:complexType>
1678 127         <xs:sequence>
1679 128             <xs:element name="EnvironmentConstraint"
1680 129                 type="tEnvironmentConstraint" maxOccurs="unbounded"/>
1681 130         </xs:sequence>
1682 131     </xs:complexType>
1683 132 </xs:element>
1684 133 <xs:element name="DeploymentArtifacts" minOccurs="0">
1685 134     <xs:complexType>
1686 135         <xs:sequence>
1687 136             <xs:element name="DeploymentArtifact"
1688 137                 type="tDeploymentArtifact" maxOccurs="unbounded"/>
1689 138         </xs:sequence>
1690 139     </xs:complexType>
1691 140 </xs:element>
1692 141 <xs:element name="ImplementationArtifacts" minOccurs="0">
1693 142     <xs:complexType>
1694 143         <xs:sequence>
1695 144             <xs:element name="ImplementationArtifact"
1696 145                 type="tImplementationArtifact" maxOccurs="unbounded"/>
1697 146         </xs:sequence>
1698 147     </xs:complexType>
1699 148 </xs:element>
1700 149 </xs:sequence>
1701 150 <xs:attribute name="id" type="xs:ID" use="required"/>
1702 151 <xs:attribute name="name" type="xs:string" use="optional"/>
1703 152 <xs:attribute name="nodeType" type="xs:QName" use="required"/>
1704 153 <xs:attribute name="minInstances" type="xs:int" use="optional"
1705 154     default="1"/>
1706 155 <xs:attribute name="maxInstances" use="optional" default="1">
1707 156     <xs:simpleType>
1708 157         <xs:union>
1709 158             <xs:simpleType>
1710 159                 <xs:restriction base="xs:nonNegativeInteger">
1711 160                     <xs:pattern value="([1-9]+[0-9]*)"/>
1712 161                 </xs:restriction>
1713 162             </xs:simpleType>
1714 163             <xs:simpleType>
1715 164                 <xs:restriction base="xs:string">
1716 165                     <xs:enumeration value="unbounded"/>
1717 166                 </xs:restriction>

```



```

1718 167         </xs:simpleType>
1719 168         </xs:union>
1720 169         </xs:simpleType>
1721 170         </xs:attribute>
1722 171         </xs:extension>
1723 172         </xs:complexContent>
1724 173     </xs:complexType>
1725 174
1726 175     <xs:complexType name="tPropertyConstraint">
1727 176         <xs:sequence>
1728 177             <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
1729 178         </xs:sequence>
1730 179         <xs:attribute name="property" type="xs:string" use="required"/>
1731 180         <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
1732 181     </xs:complexType>
1733 182
1734 183     <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1735 184     <xs:complexType name="tTopologyTemplate">
1736 185         <xs:complexContent>
1737 186             <xs:extension base="tTopologyElementCollection"/>
1738 187         </xs:complexContent>
1739 188     </xs:complexType>
1740 189
1741 190     <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1742 191     <xs:complexType name="tGroupTemplate">
1743 192         <xs:complexContent>
1744 193             <xs:extension base="tTopologyElementCollection">
1745 194                 <xs:sequence>
1746 195                     <xs:element name="Policies" minOccurs="0">
1747 196                         <xs:complexType>
1748 197                             <xs:sequence>
1749 198                                 <xs:element name="Policy" type="tPolicy"
1750 199                                     maxOccurs="unbounded"/>
1751 200                             </xs:sequence>
1752 201                         </xs:complexType>
1753 202                     </xs:element>
1754 203                 </xs:sequence>
1755 204                 <xs:attribute name="minInstances" type="xs:int" use="optional"
1756 205                     default="1"/>
1757 206                 <xs:attribute name="maxInstances" use="optional" default="1">
1758 207                     <xs:simpleType>
1759 208                         <xs:union>
1760 209                             <xs:simpleType>
1761 210                                 <xs:restriction base="xs:nonNegativeInteger">
1762 211                                     <xs:pattern value="([1-9]+[0-9]*)"/>
1763 212                                 </xs:restriction>
1764 213                             </xs:simpleType>
1765 214                             <xs:simpleType>
1766 215                                 <xs:restriction base="xs:string">
1767 216                                     <xs:enumeration value="unbounded"/>
1768 217                                 </xs:restriction>
1769 218                             </xs:simpleType>
1770 219                         </xs:union>
1771 220                     </xs:simpleType>
1772 221                 </xs:attribute>
1773 222             </xs:extension>
1774 223         </xs:complexContent>

```

```

1775 224     </xs:complexType>
1776 225
1777 226     <xs:complexType name="tTopologyElementCollection">
1778 227         <xs:complexContent>
1779 228             <xs:extension base="tExtensibleElements">
1780 229                 <xs:choice maxOccurs="unbounded">
1781 230                     <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1782 231                     <xs:element name="RelationshipTemplate"
1783 232                         type="tRelationshipTemplate"/>
1784 233                     <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1785 234                 </xs:choice>
1786 235                 <xs:attribute name="id" type="xs:ID" use="required"/>
1787 236                 <xs:attribute name="name" type="xs:string" use="optional"/>
1788 237                 <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1789 238             </xs:extension>
1790 239         </xs:complexContent>
1791 240     </xs:complexType>
1792 241
1793 242     <xs:element name="RelationshipTypes" type="tRelationshipTypes"/>
1794 243     <xs:complexType name="tRelationshipTypes">
1795 244         <xs:sequence>
1796 245             <xs:element name="RelationshipType" type="tRelationshipType"
1797 246                 maxOccurs="unbounded"/>
1798 247         </xs:sequence>
1799 248         <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1800 249     </xs:complexType>
1801 250
1802 251     <xs:element name="RelationshipType" type="tRelationshipType"/>
1803 252     <xs:complexType name="tRelationshipType">
1804 253         <xs:complexContent>
1805 254             <xs:extension base="tExtensibleElements">
1806 255                 <xs:sequence>
1807 256                     <xs:element name="RelationshipTypeProperties" minOccurs="0">
1808 257                         <xs:complexType>
1809 258                             <xs:attribute name="element" type="xs:QName"/>
1810 259                             <xs:attribute name="type" type="xs:QName"/>
1811 260                         </xs:complexType>
1812 261                     </xs:element>
1813 262                     <xs:element name="InstanceStates"
1814 263                         type="tTopologyElementInstanceStates" minOccurs="0"/>
1815 264                     </xs:sequence>
1816 265                     <xs:attribute name="id" type="xs:ID" use="required"/>
1817 266                     <xs:attribute name="name" type="xs:string" use="optional"/>
1818 267                     <xs:attribute name="semantics" type="xs:anyURI" use="required"/>
1819 268                     <xs:attribute name="cascadingDeletion" type="tBoolean"
1820 269                         use="optional" default="no"/>
1821 270                     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1822 271                 </xs:extension>
1823 272             </xs:complexContent>
1824 273         </xs:complexType>
1825 274
1826 275     <xs:element name="RelationshipTemplate" type="tRelationshipTemplate"/>
1827 276     <xs:complexType name="tRelationshipTemplate">
1828 277         <xs:complexContent>
1829 278             <xs:extension base="tExtensibleElements">
1830 279                 <xs:sequence>
1831 280                     <xs:element name="SourceElement">

```

```

1832 281      <xs:complexType>
1833 282      <xs:attribute name="id" type="xs:IDREF" use="required"/>
1834 283      </xs:complexType>
1835 284    </xs:element>
1836 285    <xs:choice>
1837 286      <xs:element name="TargetElement">
1838 287        <xs:complexType>
1839 288          <xs:attribute name="id" type="xs:IDREF" use="required"/>
1840 289        </xs:complexType>
1841 290      </xs:element>
1842 291      <xs:element name="TargetElementReference">
1843 292        <xs:complexType>
1844 293          <xs:attribute name="id" type="xs:QName" use="required"/>
1845 294        </xs:complexType>
1846 295      </xs:element>
1847 296    </xs:choice>
1848 297    <xs:element name="PropertyDefaults" minOccurs="0">
1849 298      <xs:complexType>
1850 299        <xs:sequence>
1851 300          <xs:any namespace="##other" processContents="lax"/>
1852 301        </xs:sequence>
1853 302      </xs:complexType>
1854 303    </xs:element>
1855 304    <xs:element name="PropertyConstraints" minOccurs="0">
1856 305      <xs:complexType>
1857 306        <xs:sequence>
1858 307          <xs:element name="PropertyConstraint"
1859 308            type="tPropertyConstraint" maxOccurs="unbounded"/>
1860 309        </xs:sequence>
1861 310      </xs:complexType>
1862 311    </xs:element>
1863 312    <xs:element name="RelationshipConstraints" minOccurs="0">
1864 313      <xs:complexType>
1865 314        <xs:sequence>
1866 315          <xs:element name="RelationshipConstraint"
1867 316            maxOccurs="unbounded">
1868 317            <xs:complexType>
1869 318              <xs:sequence>
1870 319                <xs:any namespace="##other" processContents="lax"
1871 320                  minOccurs="0"/>
1872 321              </xs:sequence>
1873 322              <xs:attribute name="constraintType" type="xs:anyURI"
1874 323                use="required"/>
1875 324            </xs:complexType>
1876 325          </xs:element>
1877 326        </xs:sequence>
1878 327      </xs:complexType>
1879 328    </xs:element>
1880 329  </xs:sequence>
1881 330  <xs:attribute name="id" type="xs:ID" use="required"/>
1882 331  <xs:attribute name="name" type="xs:string" use="optional"/>
1883 332  <xs:attribute name="relationshipType" type="xs:QName"
1884 333    use="required"/>
1885 334  </xs:extension>
1886 335  </xs:complexContent>
1887 336  </xs:complexType>
1888 337

```

```

1889 338 <xs:element name="NodeTypes" type="tNodeTypes"/>
1890 339 <xs:complexType name="tNodeTypes">
1891 340 <xs:sequence>
1892 341 <xs:element name="NodeType" type="tNodeType" maxOccurs="unbounded"/>
1893 342 </xs:sequence>
1894 343 <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1895 344 </xs:complexType>
1896 345
1897 346 <xs:element name="NodeType" type="tNodeType"/>
1898 347 <xs:complexType name="tNodeType">
1899 348 <xs:complexContent>
1900 349 <xs:extension base="tExtensibleElements">
1901 350 <xs:sequence>
1902 351 <xs:element name="NodeTypeProperties" minOccurs="0">
1903 352 <xs:complexType>
1904 353 <xs:attribute name="element" type="xs:QName"/>
1905 354 <xs:attribute name="type" type="xs:QName"/>
1906 355 </xs:complexType>
1907 356 </xs:element>
1908 357 <xs:element name="DerivedFrom" minOccurs="0">
1909 358 <xs:complexType>
1910 359 <xs:attribute name="nodeTypeRef" type="xs:QName"
1911 360 use="required"/>
1912 361 </xs:complexType>
1913 362 </xs:element>
1914 363 <xs:element name="InstanceStates"
1915 364 type="tTopologyElementInstanceStates" minOccurs="0"/>
1916 365 <xs:element name="Interfaces" minOccurs="0">
1917 366 <xs:complexType>
1918 367 <xs:sequence>
1919 368 <xs:element name="Interface" maxOccurs="unbounded">
1920 369 <xs:complexType>
1921 370 <xs:sequence>
1922 371 <xs:element name="Operation" type="tOperation"
1923 372 maxOccurs="unbounded"/>
1924 373 <xs:element name="ImplementationArtifacts" minOccurs="0">
1925 374 <xs:complexType>
1926 375 <xs:sequence>
1927 376 <xs:element name="ImplementationArtifact"
1928 377 type="tImplementationArtifact" maxOccurs="unbounded"/>
1929 378 </xs:sequence>
1930 379 </xs:complexType>
1931 380 </xs:element>
1932 381 </xs:sequence>
1933 382 <xs:attribute name="name" type="xs:anyURI" use="required"/>
1934 383 </xs:complexType>
1935 384 </xs:element>
1936 385 </xs:sequence>
1937 386 </xs:complexType>
1938 387 </xs:element>
1939 388 <xs:element name="Policies" minOccurs="0">
1940 389 <xs:complexType>
1941 390 <xs:sequence>
1942 391 <xs:element name="Policy" type="tPolicy"
1943 392 maxOccurs="unbounded"/>
1944 393 </xs:sequence>
1945 394 </xs:complexType>

```

```

1946 395     </xs:element>
1947 396     <xs:element name="DeploymentArtifacts" minOccurs="0">
1948 397         <xs:complexType>
1949 398             <xs:sequence>
1950 399                 <xs:element name="DeploymentArtifact"
1951 400                     type="tDeploymentArtifact" maxOccurs="unbounded"/>
1952 401             </xs:sequence>
1953 402         </xs:complexType>
1954 403     </xs:element>
1955 404 </xs:sequence>
1956 405 <xs:attribute name="id" type="xs:ID" use="required"/>
1957 406 <xs:attribute name="name" type="xs:string" use="optional"/>
1958 407 <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1959 408 </xs:extension>
1960 409 </xs:complexContent>
1961 410 </xs:complexType>
1962 411
1963 412 <xs:element name="Plans" type="tPlans"/>
1964 413 <xs:complexType name="tPlans">
1965 414     <xs:sequence>
1966 415         <xs:element name="Plan" type="tPlan" maxOccurs="unbounded"/>
1967 416     </xs:sequence>
1968 417     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1969 418 </xs:complexType>
1970 419
1971 420 <xs:element name="Plan" type="tPlan"/>
1972 421 <xs:complexType name="tPlan">
1973 422     <xs:complexContent>
1974 423         <xs:extension base="tExtensibleElements">
1975 424             <xs:sequence>
1976 425                 <xs:element name="Precondition" type="tCondition" minOccurs="0"/>
1977 426                 <xs:choice>
1978 427                     <xs:element name="PlanModel">
1979 428                         <xs:complexType>
1980 429                             <xs:sequence>
1981 430                                 <xs:any namespace="##other" processContents="lax"/>
1982 431                             </xs:sequence>
1983 432                         </xs:complexType>
1984 433                     </xs:element>
1985 434                     <xs:element name="PlanModelReference">
1986 435                         <xs:complexType>
1987 436                             <xs:attribute name="reference" type="xs:anyURI"
1988 437                                 use="required"/>
1989 438                         </xs:complexType>
1990 439                     </xs:element>
1991 440                 </xs:choice>
1992 441             </xs:sequence>
1993 442             <xs:attribute name="id" type="xs:ID" use="required"/>
1994 443             <xs:attribute name="name" type="xs:string" use="optional"/>
1995 444             <xs:attribute name="planType" type="xs:anyURI" use="required"/>
1996 445             <xs:attribute name="languageUsed" type="xs:anyURI" use="required"/>
1997 446         </xs:extension>
1998 447     </xs:complexContent>
1999 448 </xs:complexType>
2000 449
2001 450 <xs:complexType name="tPolicy">
2002 451     <xs:complexContent>

```

```

2003 452     <xs:extension base="tExtensibleElements">
2004 453         <xs:attribute name="name" type="xs:string" use="required"/>
2005 454         <xs:attribute name="type" type="xs:anyURI" use="required"/>
2006 455     </xs:extension>
2007 456 </xs:complexContent>
2008 457 </xs:complexType>
2009 458
2010 459 <xs:complexType name="tEnvironmentConstraint">
2011 460     <xs:sequence>
2012 461         <xs:any namespace="##other" processContents="lax"/>
2013 462     </xs:sequence>
2014 463     <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
2015 464 </xs:complexType>
2016 465
2017 466 <xs:complexType name="tExtensions">
2018 467     <xs:complexContent>
2019 468         <xs:extension base="tExtensibleElements">
2020 469             <xs:sequence>
2021 470                 <xs:element name="Extension" type="tExtension"
2022 471                     maxOccurs="unbounded"/>
2023 472             </xs:sequence>
2024 473         </xs:extension>
2025 474     </xs:complexContent>
2026 475 </xs:complexType>
2027 476
2028 477 <xs:complexType name="tExtension">
2029 478     <xs:complexContent>
2030 479         <xs:extension base="tExtensibleElements">
2031 480             <xs:attribute name="namespace" type="xs:anyURI" use="required"/>
2032 481             <xs:attribute name="mustUnderstand" type="tBoolean" use="optional"
2033 482                 default="yes"/>
2034 483         </xs:extension>
2035 484     </xs:complexContent>
2036 485 </xs:complexType>
2037 486
2038 487 <xs:complexType name="tParameter">
2039 488     <xs:attribute name="name" type="xs:string" use="required"/>
2040 489     <xs:attribute name="type" type="xs:string" use="required"/>
2041 490     <xs:attribute name="required" type="tBoolean" use="optional"
2042 491         default="yes"/>
2043 492 </xs:complexType>
2044 493
2045 494 <xs:complexType name="tWSDL">
2046 495     <xs:attribute name="portType" type="xs:QName" use="required"/>
2047 496     <xs:attribute name="operation" type="xs:NCName" use="required"/>
2048 497 </xs:complexType>
2049 498
2050 499 <xs:complexType name="tOperation">
2051 500     <xs:complexContent>
2052 501         <xs:extension base="tExtensibleElements">
2053 502             <xs:choice>
2054 503                 <xs:element name="WSDL" type="tWSDL"/>
2055 504                 <xs:element name="REST" type="tREST"/>
2056 505                 <xs:element name="ScriptOperation" type="tScriptOperation"/>
2057 506             </xs:choice>
2058 507             <xs:attribute name="name" type="xs:NCName" use="required"/>
2059 508         </xs:extension>

```

```

2060 509     </xs:complexContent>
2061 510 </xs:complexType>
2062 511
2063 512 <xs:complexType name="tREST">
2064 513   <xs:sequence>
2065 514     <xs:element name="Parameters" minOccurs="0">
2066 515       <xs:complexType>
2067 516         <xs:sequence>
2068 517           <xs:element name="Parameter" maxOccurs="unbounded">
2069 518             <xs:complexType>
2070 519               <xs:attribute name="name" type="xs:string" use="required"/>
2071 520               <xs:attribute name="required" type="tBoolean" use="optional"
2072 521                 default="yes"/>
2073 522             </xs:complexType>
2074 523           </xs:element>
2075 524         </xs:sequence>
2076 525       </xs:complexType>
2077 526     </xs:element>
2078 527     <xs:element name="Headers" minOccurs="0">
2079 528       <xs:complexType>
2080 529         <xs:sequence>
2081 530           <xs:element name="Header" maxOccurs="unbounded">
2082 531             <xs:complexType>
2083 532               <xs:attribute name="name" type="xs:string" use="required"/>
2084 533               <xs:attribute name="required" type="tBoolean" use="optional"
2085 534                 default="yes"/>
2086 535             </xs:complexType>
2087 536           </xs:element>
2088 537         </xs:sequence>
2089 538       </xs:complexType>
2090 539     </xs:element>
2091 540   </xs:sequence>
2092 541   <xs:attribute name="method" default="GET">
2093 542     <xs:simpleType>
2094 543       <xs:restriction base="xs:string">
2095 544         <xs:enumeration value="GET"/>
2096 545         <xs:enumeration value="PUT"/>
2097 546         <xs:enumeration value="POST"/>
2098 547         <xs:enumeration value="DELETE"/>
2099 548       </xs:restriction>
2100 549     </xs:simpleType>
2101 550   </xs:attribute>
2102 551   <xs:attribute name="abs_path" type="xs:anyURI" use="optional"/>
2103 552   <xs:attribute name="absoluteURI" type="xs:anyURI" use="optional"/>
2104 553   <xs:attribute name="requestBody" type="xs:QName" use="optional"/>
2105 554   <xs:attribute name="responseBody" type="xs:QName" use="optional"/>
2106 555 </xs:complexType>
2107 556
2108 557 <xs:complexType name="tScriptOperation">
2109 558   <xs:sequence>
2110 559     <xs:element name="InputParameters" minOccurs="0">
2111 560       <xs:complexType>
2112 561         <xs:sequence>
2113 562           <xs:element name="InputParameter" type="tParameter"
2114 563             maxOccurs="unbounded"/>
2115 564         </xs:sequence>
2116 565       </xs:complexType>

```

```

2117 566     </xs:element>
2118 567     <xs:element name="OutputParameters" minOccurs="0">
2119 568         <xs:complexType>
2120 569             <xs:sequence>
2121 570                 <xs:element name="OutputParameter" type="tParameter"
2122 571                     maxOccurs="unbounded"/>
2123 572             </xs:sequence>
2124 573         </xs:complexType>
2125 574     </xs:element>
2126 575 </xs:sequence>
2127 576 </xs:complexType>
2128 577 <xs:complexType name="tImplementationArtifact">
2129 578     <xs:complexContent>
2130 579         <xs:extension base="tExtensibleElements">
2131 580             <xs:sequence>
2132 581                 <xs:element name="RequiredContainerCapabilities" minOccurs="0">
2133 582                     <xs:complexType>
2134 583                         <xs:sequence>
2135 584                             <xs:element name="RequiredContainerCapability"
2136 585                                 maxOccurs="unbounded">
2137 586                                     <xs:complexType>
2138 587                                         <xs:attribute name="capability" type="xs:anyURI"
2139 588                                             use="required"/>
2140 589                                     </xs:complexType>
2141 590                                 </xs:element>
2142 591                             </xs:sequence>
2143 592                         </xs:complexType>
2144 593                     </xs:element>
2145 594                 </xs:sequence>
2146 595                 <xs:attribute name="operationName" type="xs:string"
2147 596                     use="required"/>
2148 597                 <xs:attribute name="type" type="xs:anyURI" use="required"/>
2149 598             </xs:extension>
2150 599         </xs:complexContent>
2151 600     </xs:complexType>
2152 601
2153 602 <xs:complexType name="tCondition">
2154 603     <xs:sequence>
2155 604         <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2156 605     </xs:sequence>
2157 606     <xs:attribute name="expressionLanguage" type="xs:anyURI"
2158 607         use="required"/>
2159 608 </xs:complexType>
2160 609
2161 610 <xs:complexType name="tTopologyElementInstanceStates">
2162 611     <xs:sequence>
2163 612         <xs:element name="InstanceState" maxOccurs="unbounded">
2164 613             <xs:complexType>
2165 614                 <xs:attribute name="state" type="xs:anyURI" use="required"/>
2166 615             </xs:complexType>
2167 616         </xs:element>
2168 617     </xs:sequence>
2169 618 </xs:complexType>
2170 619
2171 620 <xs:simpleType name="tBoolean">
2172 621     <xs:restriction base="xs:string">
2173 622         <xs:enumeration value="yes"/>

```



```
2174 623     <xs:enumeration value="no"/>
2175 624     </xs:restriction>
2176 625 </xs:simpleType>
2177 626
2178 627 <xs:simpleType name="importedURI">
2179 628     <xs:restriction base="xs:anyURI"/>
2180 629 </xs:simpleType>
2181 630
2182 631 </xs:schema>
```

Appendix E. Sample

This appendix contains the full sample used in this specification.

E.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"/>
  </TopologyTemplate>
</ServiceTemplate>
```

```

2234     <RelationshipTemplate id="MyDeploymentRelationship"
2235                           relationshipType="deployedOn">
2236         <SourceElement id="MyApplication"/>
2237         <TargetElement id="MyAppServer"/>
2238     </RelationshipTemplate>
2239
2240 </TopologyTemplate>
2241
2242 <NodeTypes>
2243     <NodeType name="Application">
2244         <documentation xml:lang="EN">
2245             A reusable definition of a node type representing an
2246             application that can be deployed on application servers.
2247         </documentation>
2248         <NodeTypeProperties element="ApplicationProperties"/>
2249         <InstanceStates>
2250             <InstanceState state="http://www.my.com/started"/>
2251             <InstanceState state="http://www.my.com/stopped"/>
2252         </InstanceStates>
2253         <Interfaces>
2254             <Interface name="DeploymentInterface">
2255                 <Operation name="DeployApplication">
2256                     <ScriptOperation>
2257                         <InputParameters>
2258                             <InputParamter name="InstanceName"
2259                                             type="string"/>
2260                             <InputParamter name="AppServerHostname"
2261                                             type="string"/>
2262                             <InputParamter name="ContextRoot"
2263                                             type="string"/>
2264                         </InputParameters>
2265                     </ScriptOperation>
2266                 </Operation>
2267                 <ImplementationArtifacts>
2268                     <ImplementationArtifact operationName="DeployApplication"
2269                                             type="http://www.my.com/ScriptArtifact/PhythonReference">
2270                         scripts/phython/deployApplication.py
2271                     </ImplementationArtifact>
2272                 </ImplementationArtifacts>
2273             </Interface>
2274         </Interfaces>
2275     </NodeType>
2276     <NodeType name="ApplicationServer"
2277               targetNamespace="http://www.ibm.com/sample">
2278         <NodeTypeProperties element="AppServerProperties"/>
2279         <Interfaces>
2280             <Interface name="MyAppServerInterface">
2281                 <Operation name="AcquireNetworkAddress">
2282                     <WSDL portType="my:NetworkPT"
2283                           operation="AcquireNetworkAddress"/>
2284                 </Operation>
2285                 <Operation name="DeployApplicationServer">
2286                     <WSDL portType="my:AppServerPT"
2287                           operation="DeployApplicationServer"/>
2288                 </Operation>
2289                 <ImplementationArtifacts>
2290                     <ImplementationArtifact operationName="AcquireNetworkAddress"

```

```

2291         type="http://www.my.com/MyJeeArtifact/EarRef">
2292         artifacts/jee/MyEAR.ear
2293     </ImplementationArtifact>
2294     <ImplementationArtifact operationName="DeployApplicationServer"
2295         type="http://www.my.com/MyJeeArtifact/EarRef">
2296         artifacts/jee/AppServerManagement.ear
2297     </ImplementationArtifact>
2298 </ImplementationArtifacts>
2299 </Interface>
2300 </Interfaces>
2301 </NodeType>
2302 </NodeTypes>
2303
2304 <RelationshipTypes>
2305     <documentation xml:lang="EN">
2306         A reusable definition of relation that expresses deployment of
2307         an artifact on a hosting environment.
2308     </documentation>
2309     <RelationshipType name="deployedOn"
2310         semantics="www.my.com/RelSemantics/deployedOn">
2311     </RelationshipType>
2312 </RelationshipTypes>
2313
2314 <Plans>
2315     <Plan id="DeployApplication"
2316         name="Sample Application Build Plan"
2317         planType="http://docs.oasis-
2318             open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
2319         languageUsed="http://www.omg.org/spec/BPMN/2.0/">
2320
2321         <PreCondition expressionLanguage="www.my.com/text">?
2322             Run only if funding is available
2323         </PreCondition>
2324
2325     <PlanModel>
2326         <process name="DeployNewApplication" id="p1">
2327             <documentation>This process deploys a new instance of the
2328                 sample application.
2329             </documentation>
2330
2331             <task id="t1" name="CreateAccount"/>
2332
2333             <task id="t2" name="AcquireNetworkAddresses"
2334                 isSequential="false"
2335                 loopDataInput="t2Input.LoopCounter"/>
2336             <documentation>Assumption: t2 gets data of type "input"
2337                 as input and this data has a field names "LoopCounter"
2338                 that contains the actual multiplicity of the task.
2339             </documentation>
2340
2341             <task id="t3" name="DeployApplicationServer"
2342                 isSequential="false"
2343                 loopDataInput="t3Input.LoopCounter"/>
2344
2345             <task id="t4" name="DeployApplication"
2346                 isSequential="false"
2347                 loopDataInput="t4Input.LoopCounter"/>

```

```
2348
2349     <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
2350     <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
2351     <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
2352   </process>
2353 </PlanModel>
2354 </Plan>
2355
2356 <Plan id="RemoveApplication"
2357   planType="http://docs.oasis-
2358     open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
2359   languageUsed="http://docs.oasis-
2360     open.org/wsbpel/2.0/process/executable">
2361   <PlanModelReference reference="prj:RemoveApp"/>
2362 </Plan>
2363 </Plans>
2364
2365 </ServiceTemplate>
```

Appendix F. 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.
wd-04	2012-03-22	Thomas Spatzier, Frank Leymann	Changes for JIRA Issue TOSCA-4: Changed definition of <code>NodeType</code> <code>Interfaces</code> element; adapted text and examples
wd-05	2012-03-30	Thomas Spatzier, Frank Leymann	Changes for JIRA Issue TOSCA-5: Changed definition of <code>NodeTemplate</code> to include <code>ImplementationArtifact</code> element; adapted text Added Acknowledgements section in Appendix