
Topology and Orchestration Specification for Cloud Applications Version 1.0

Working Draft 03

06 March 2012

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

Declared XML namespaces:

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

Abstract:

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

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

Status:

This [Working Draft](#) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](#) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](#) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

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.

Table of Contents

1	Introduction.....	5
2	Language Design	5
2.1	Dependencies on Other Specifications	5
2.2	Notational Conventions.....	5
2.3	Normative References	5
2.4	Non-Normative References	6
2.5	Namespaces.....	6
2.6	Language Extensibility.....	6
2.7	Overall Language Structure.....	7
2.7.1	Syntax.....	7
2.7.2	Properties	7
3	Core Concepts and Usage Pattern	10
3.1	Core Concepts.....	10
3.2	Use Cases	12
3.2.1	Services as Marketable Entities	12
3.2.2	Portability of Service Templates.....	12
3.2.3	Service Composition	13
3.2.4	Relation to Virtual Images	13
4	Node Types	13
4.1	Syntax.....	13
4.2	Properties.....	15
4.3	Derivation Rules	17
4.4	Example	18
5	Relationship Types.....	19
5.1	Syntax.....	19
5.2	Properties.....	19
5.3	Example	20
6	Topology Template.....	20
6.1	Syntax.....	20
6.2	Properties.....	22
6.3	Example	26
7	Plans.....	27
7.1	Syntax.....	27
7.2	Properties.....	27
7.3	Use of Process Modeling Languages.....	28
7.4	Example	28
8	Security Considerations	29
9	Conformance	30
Appendix A.	Portability and Interoperability Considerations	31
Appendix B.	Complete TOSCA Grammar	32
Appendix C.	TOSCA Schema.....	37
Appendix D.	Sample	48
D.1	Sample Service Topology Definition	48

Appendix E. Revision History 52

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/ |

- 43 **[XML Schema Part 1]** XML Schema Part 1: Structures, W3C Recommendation, October 2004,
44 <http://www.w3.org/TR/xmlschema-1/>
- 45 **[XML Schema Part 2]** XML Schema Part 2: Datatypes, W3C Recommendation, October 2004,
46 <http://www.w3.org/TR/xmlschema-2/>
- 47 **[XMLSpec]** XML Specification, W3C Recommendation, February 1998,
48 <http://www.w3.org/TR/1998/REC-xml-19980210>
- 49 **[XPath 1.0]** XML Path Language (XPath) Version 1.0, W3C Recommendation, November
50 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>
- 51

52 **2.4 Non-Normative References**

- 53 **[Reference]** [Full reference citation]
- 54

55 **2.5 Namespaces**

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

61

Prefix	Namespace
ste	http://docs.oasis-open.org/tosca/ns/2011/12
xs	http://www.w3.org/2001/XMLSchema
wSDL	http://schemas.xmlsoap.org/wSDL/
bpmn	http://www.omg.org/bpmn/2.0

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

63

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

67 **2.6 Language Extensibility**

68 The TOSCA extensibility mechanism allows:

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

73 The specification differentiates between mandatory and optional extensions (the section below explains
74 the syntax used to declare extensions). If a mandatory extension is used, a compliant implementation

75 MUST understand the extension. If an optional extension is used, a compliant implementation MAY
76 ignore the extension.

77 2.7 Overall Language Structure

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

82 2.7.1 Syntax

```
83 1 <ServiceTemplate id="ID"  
84 2     name="string"?  
85 3     targetNamespace="anyURI">  
86 4  
87 5     <Extensions>?  
88 6         <Extension namespace="anyURI"  
89 7             mustUnderstand="yes|no"?/>>+  
90 8     </Extensions>  
91 9  
92 10    <Import namespace="anyURI"?  
93 11        location="anyURI"?  
94 12        importType="anyURI"/>*  
95 13  
96 14    <Types>?  
97 15        <xs:schema .../>*  
98 16    </Types>  
99 17  
100 18    (  
101 19        <TopologyTemplate>  
102 20            ...  
103 21        </TopologyTemplate>  
104 22    |  
105 23        <TopologyTemplateReference reference="xs:QName">  
106 24    )?  
107 25  
108 26    <NodeTypes>?  
109 27        ...  
110 28    </NodeTypes>  
111 29  
112 30    <RelationshipTypes>?  
113 31        ...  
114 32    </RelationshipTypes>  
115 33  
116 34    <Plans>?  
117 35        ...  
118 36    </Plans>  
119 37  
120 38 </ServiceTemplate>
```

121 2.7.2 Properties

122 The `ServiceTemplate` element has the following properties:

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

125

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

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

130

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

- 133 • `targetNamespace`: The value of this attribute is the namespace for the Service Template.

- 134 • `Extensions`: This element specifies namespaces of TOSCA extension attributes and extension
135 elements. The element is optional.

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

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

145 The same extension URI MAY be declared multiple times in the `Extensions` element. If an
146 extension URI is identified as mandatory in one `Extension` element and optional in another,
147 then the mandatory semantics have precedence and MUST be enforced. The extension
148 declarations in an `Extensions` element MUST be treated as an unordered set.

- 149 • `Import`: This element declares a dependency on external Service Template, XML Schema
150 definitions, or WSDL definitions. Any number of `Import` elements MAY appear as children of the
151 `ServiceTemplate` element.

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

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

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

172 According to these rules, it is permissible to have an `Import` element without `namespace` and
173 `location` attributes, and only containing an `importType` attribute. Such an `Import` element

174 indicates that external definitions of the indicated type are in use that are not namespace-
175 qualified, and makes no statement about where those definitions might be found.

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

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

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

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

200 • `TopologyTemplate`: This element specifies in place the topological structure of an IT service by
201 means of a directed graph.

202
203 The main ingredients of a Topology Template are a set of Node Templates and Relationship
204 Templates. The Node Templates are the nodes of the directed graph. The Relationship
205 Templates are the directed edges between the nodes; each indicates the semantics of the
206 corresponding relationships.

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

211
212 Note that either zero or one Topology Template MUST occur in a Service Template, either
213 defined in place via a `TopologyTemplate` element or referenced via a
214 `TopologyTemplateReference`.

215 • `NodeTypes`: This element specifies the types of Node (Templates), i.e., their properties and
216 behavior.

217 • `RelationshipTypes`: This element specifies the types of relationships, i.e. the kind of links
218 between Node Templates within a Service Template, and their properties.

219 • `Plans`: This element specifies the operational behavior of the service. Each `Plan` contained in
220 the `Plans` element specifies how to create, terminate or manage the service.

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

230 Example of the use of a type definition:

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

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

```
249 1 <documentation source="anyURI"? xml:lang="language"?>  
250 2   ...  
251 3 </documentation>
```

252 Example of use of a documentation:

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

261 3 Core Concepts and Usage Pattern

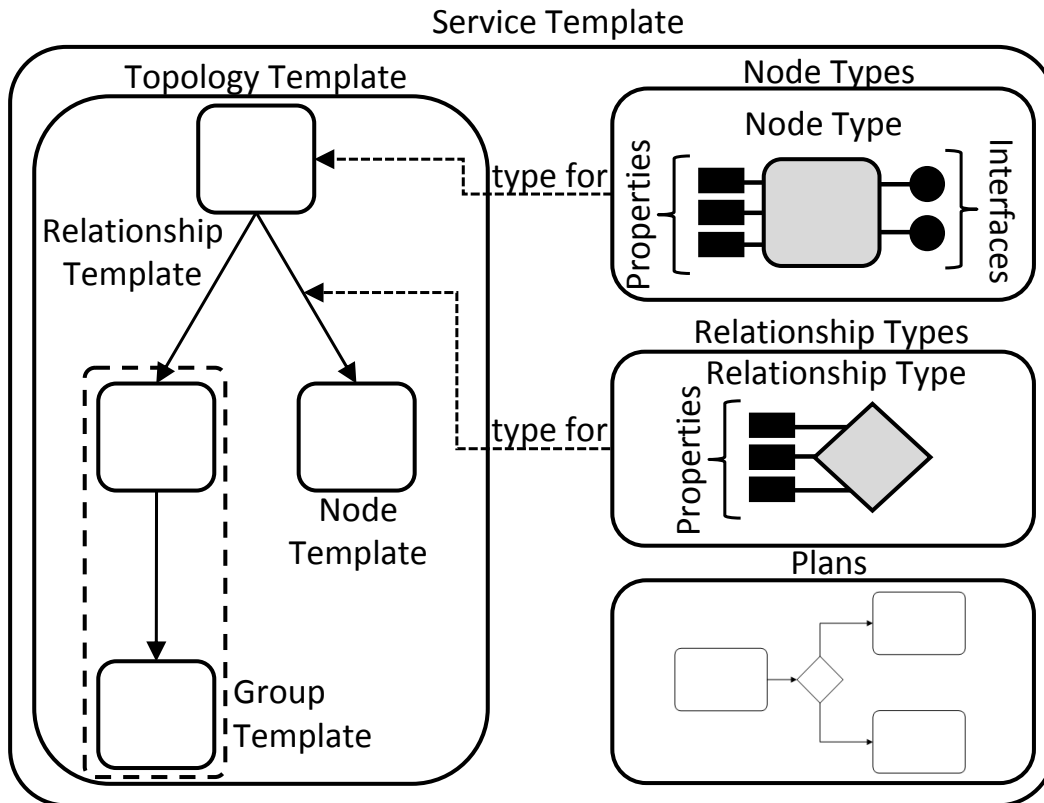
262 The main concepts behind TOSCA are described and some usage patterns of Service Templates are
263 sketched.

264 3.1 Core Concepts

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

270

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



278
 279 *Figure 1: Structural Elements of a Service Template and their Relations*

280 For example, consider a service that consists of an application server, a process engine, and a process
 281 model. A Topology Template defining that service would include one Node Template of Node Type
 282 "application server", another Node Template of Node Type "process engine", and a third Node Template
 283 of Node Type "process model". The application server Node Type defines properties like the IP address
 284 of an instance of this type, an operation for installing the application server with the corresponding IP
 285 address, and an operation for shutting down an instance of this application server. A constraint in the
 286 Node Template can specify a range of IP addresses available when making a concrete application server
 287 available.

288 A *Relationship Template* specifies the occurrence of a relationship between nodes in a Topology
 289 Template. Each Relationship Template refers to a Relationship type that defines the semantics and any
 290 properties of the relationship. Relationship Types are defined separately for reuse purposes. The
 291 Relationship Template indicates the types of nodes and the direction of the relationship by defining one
 292 source and one target Node Template (in nested SourceNodeTemplate and TargetNodeTemplate
 293 elements). The Relationship Template also defines any constraints with the optional
 294 RelationshipConstraints element.

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

298 A deployed service is an instance of a Service Template. More precisely, the instance is derived by
 299 instantiating the Topology Template of its Service Template, most often by running a special plan defined

300 for the Service Template, often referred to as build plan. The build plan will provide actual values for the
301 various properties of the various Node Templates and Relationship Templates of the Topology Template.
302 These values can come from input passed in by users as triggered by human interactions defined within
303 the build plan, by automated operations defined within the build plan (such as a directory lookup), or the
304 templates can specify default values for some properties. The build plan will typically make use of
305 operations of the Node Types of the Node Templates.

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

312 *Plans* defined in a Service Template describe the management aspects of service instances, especially
313 their creation and termination. These plans are defined as process models, i.e. a workflow of one or more
314 steps. Instead of providing another language for defining process models, the specification relies on
315 existing languages like BPMN or BPEL. Relying on existing standards in this space facilitates portability
316 and interoperability, but any language for defining process models can be used. The TOSCA metamodel
317 provides containers to either refer to a process model (via *Plan Model Reference*) or to include the actual
318 model in the plan (via *Plan Model*). A process model can contain tasks (using BPMN terminology) that
319 refer to operations of Interfaces of Node Templates or any other interface (e.g. the invocation of an
320 external service for licensing); in doing so, a plan can directly manipulate nodes of the topology of a
321 service or interact with external systems.

322 **3.2 Use Cases**

323 The specification supports at least the following major use cases.

324 **3.2.1 Services as Marketable Entities**

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

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

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

346 **3.2.2 Portability of Service Templates**

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

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

355 3.2.3 Service Composition

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

362 3.2.4 Relation to Virtual Images

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

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

375 4 Node Types

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

382 4.1 Syntax

```
383 1 <NodeTypes>?  
384 2  
385 3 <NodeType id="ID"  
386 4     name="string"?>+  
387 5  
388 6 <NodeTypeProperties element="QName"?  
389 7     type="QName"?/>?  
390 8  
391 9 <DerivedFrom nodeTypeRef="QName"/>?  
392 10  
393 11 <InstanceStates>?  
394 12 <InstanceState state="anyURI">+  
395 13 </InstanceStates>  
396 14  
397 15 <Interfaces>?  
398 16
```

```

399 17     <Interface>+
400 18
401 19     (
402 20         <WSDL portType="QName"
403 21             operation="NCName"?>+
404 22     |
405 23         <REST method="GET | PUT | POST | DELETE"
406 24             requestURI="anyURI"
407 25             requestPayload="QName"?
408 26             responsePayload="QName"?>+
409 27     |
410 28         <Operation name="NCame">+
411 29
412 30         <InputParameters>?
413 31
414 32             <InputParamter name="string"
415 33                 type="string"
416 34                 required="yes|no">+
417 35
418 36         </InputParameters>
419 37
420 38         <OutputParameters>?
421 39
422 40             <OutputParamter name="string"
423 41                 type="string"
424 42                 required="yes|no">+
425 43
426 44         </OutputParameters>
427 45
428 46         <Implementations>
429 47
430 48             <Implementation implementationID="anyURI"?
431 49                 language="anyURI"?>+
432 50         (
433 51             <ImplementationProper>?
434 52                 code
435 53             </ImplementationProper>
436 54         |
437 55             <ImplementationReference ref="anyURI"/>?
438 56         )
439 57         <Implementation>
440 58
441 59         </Implementations>
442 60     </Operation>
443 61 )
444 62
445 63 </Interface>
446 64
447 65 </Interfaces>
448 66
449 67 <Policies>?
450 68     <Policy name="string" type="anyURI">+
451 69         policy specific content
452 70     </Policy>
453 71 </Policies>
454 72
455 73 <DeploymentArtifacts>?
456 74     <DeploymentArtifact name="string" type="anyURI">+

```

```
457 75         artifact specific content
458 76     </DeploymentArtifact>
459 77 </DeploymentArtifacts>
460 78
461 79 </NodeType>
462 80
463 81 </NodeTypes>
```

4.2 Properties

464 The `NodeType` element has the following properties:

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

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

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

489 The `DerivedFrom` element has the following properties:

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

492 The `InstanceStates` element has the following properties:

- 493 • `InstanceState`: specifies a potential state.

494 The `InstanceState` element has the following properties:

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

496 The `Interface` element has one of the following properties:

- 497 • `WSDL`: specifies a WSDL port type or an operation of a port type as part of the Interface.
- 498 • `REST`: specifies an HTTP request as part of the Interface.

- `Operation`: specifies a proprietary implementation as part of the Interface.

The `WSDL` element has the following properties:

- `portType`: This is the QName of the port type that contains the definition of one or more operations defined as part of the Interface. Note that the namespace of the portType MUST be imported.
- `operation`: This optional attribute specifies the name of a single operation of the port type to become part of the Node Type Interface. If this attribute is not specified, the complete WSDL port type becomes part of the Node Type Interface.

The `REST` element has the following properties:

- `method`: This is the name of an HTTP method (often in a REST-style Interface).
- `requestURI`: This is the requestURI necessary to create the request.
- `requestPayload`: The QName specifies the schema of the HTTP message payload passed with the request to the HTTP processor.
- `responsePayload`: The QName specifies the schema of the payload passed with the response message from the HTTP processor.

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

The `Operation` element has the following properties:

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

The `Policy` element has the following properties:

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

Consider a hypothetical billing policy. In this example the type `www.sample.com/BillingPractice` could define a policy for billing usage of a service instance. The policy specific content can define the interface providing the operations to perform billing. Further content could specify the granularity of the base for payment, e.g. it could provide an enumeration with the possible values

543 “service”, “resource”, and “labor”. A value of “service” might specify that an instance of the
544 corresponding node will be billed during its instance lifetime. A value of “resource” might specify
545 that the resources consumed by an instance will be billed. A value of “labor” might specify that the
546 use of a plan affecting a node instance will be billed.

547 The `DeploymentArtifact` element has the following properties:

- 548 • `name`: The attribute specifies the name of the artifact. Note, that uniqueness of the name within
549 the scope of the encompassing Node Type SHOULD be guaranteed by the definition.
- 550 • `type`: The attribute specifies the type of the deployment artifact definition that is related to the
551 Node Type, i.e. the attribute gives a hint how to interpret the body of the `DeploymentArtifact`
552 element.

553 Note, that the combination of name and type SHOULD be unique within the scope of the Node
554 Type.

- 555 • The body of this element contains the type-specific content.

556
557 For example, if the `type` attribute contains the value
558 <http://docs.oasis-open.org/tosca/ns/2011/12/deploymentArtifacts/ovfRef>, the body will contain an
559 XML fragment with a reference to an OVF package and a mapping between service template
560 data and elements of the respective OVF envelope.

561 4.3 Derivation Rules

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

- 563 • Node Type Properties: It is assumed that the XML element (or type) representing the Node Type
564 Properties extends the XML element (or type) of the Node Type Properties of the Node Type
565 referenced in the `DerivedFrom` element.
- 566 • Instance States: The set of instance states of this Node Type consists of the set union of the
567 instances states defined by the Nodes Type derived from and the instance states defined by this
568 Node Type. A set of instance states of the same name will be combined into a single instance
569 state of the same name.
- 570 • WSDL: The set of WSDL operations of this Node Type consists of the set union of the WSDL
571 operations defined by the Node Type derived from and the WSDL operations defined by the Node
572 Type. A WSDL operation defined by this Node Type substitutes a WSDL operation with the same
573 name and signature of the Node Type derived from.
- 574 • REST: The set of REST requests of this Node Type consists of the set union of the REST
575 requests defined by the Nodes Type derived from and the REST requests defined by this Node
576 Type. A REST request defined by this Node Type substitutes a REST request with the same
577 identity of the Node Type derived from.
- 578 • Operation: The set of operations of this Node Type consists of the set union of the operations
579 defined by the Nodes Type derived from and the operations defined by this Node Type. An
580 operation defined by this Node Type substitutes an operation with the same name of the Node
581 Type derived from.
- 582 • Deployment Artifacts: The set of deployment artifacts of this Node Type consists of the set union
583 of the deployment artifacts defined by the Nodes Type derived from and the deployment artifacts
584 defined by this Node Type. A deployment artifact defined by this Node Type substitutes a
585 deployment artifact with the same name and type of the Node Type derived from.

- 586
- Policies: The set of policies of this Node Type consists of the set union of the policies defined by the Nodes Type derived from and the policies defined by this Node Type. A policy defined by this Node Type substitutes a policy with the same name and type of the Node Type derived from.
- 587
- 588

589 4.4 Example

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

```
594 <ServiceTemplate id="myService" name="My Service"  
595           targetNamespace="http://www.ibm.com/sample">  
596  
597   <NodeTypes>  
598  
599     <NodeType id="Project" name="My Project">  
600  
601       <documentation xml:lang="EN">  
602         A reusable definition of a node type supporting  
603         the creation of new projects.  
604       </documentation>  
605  
606       <NodeTypeProperties element="ProjectProperties"/>  
607  
608       <InstanceStates>  
609         <InstanceState state="www.my.com/active"/>  
610         <InstanceState state="www.my.com/onHalt"/>  
611       </InstanceStates>  
612  
613       <Interfaces>  
614         <Interface>  
615           <Operation name="CreateProject">  
616             <InputParameters>  
617               <InputParamter name="ProjectName"  
618                 type="string"/>  
619               <InputParamter name="Owner"  
620                 type="string"/>  
621               <InputParamter name="AccountID"  
622                 type="string"/>  
623             </InputParameters>  
624             <Implementations>  
625               <Implementation>  
626                 ...  
627               </Implementation>  
628             </Implementations>  
629           </Operation>  
630         </Interface>  
631       </Interfaces>  
632     </NodeType>  
633   </NodeTypes>  
634 </ServiceTemplate>
```

638 The Node Type “Project” has three Node Type Properties defined as an XML element in the `Types`
639 element definition of the Service Template document: `Owner`, `ProjectName` and `AccountID` which are all

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

647 5 Relationship Types

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

652 5.1 Syntax

```
653 1 <RelationshipTypes>  
654 2  
655 3   <RelationshipType id="ID"  
656 4     name="string"?  
657 5     semantics="anyURI"  
658 6     cascadingDeletion="yes|no"?>+  
659 7  
660 8     <RelationshipTypeProperties element="QName"?  
661 9       type="QName"?/>?  
662 10  
663 11   <InstanceStates>?  
664 12     <InstanceState state="anyURI">+  
665 13   </InstanceStates>  
666 14  
667 15 </RelationshipType>  
668 16  
669 17 </RelationshipTypes>
```

670 5.2 Properties

671 The `RelationshipType` element has the following properties:

- 672 • `id`: This attribute specifies the identifier of the Relationship Type. The identifier of the
673 Relationship Type MUST be unique within the target namespace.
- 674 • `name`: This optional attribute specifies the name of the Relationship Type.
- 675 • `semantics`: The meaning or expected behavior of an instance of this Relationship Type.
- 676 • `cascadingDeletion`: If set to "yes" the target of an instance of a Relationship Template of this
677 RelationshipType is automatically deleted when the source of the instance of the Relationship
678 Template is deleted.

679 The `RelationshipTypeProperties` element has the following properties:

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

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

685 The InstanceStates element has the following properties:

- 686 • InstanceState: specifies a potential state.

687 The InstanceState element has the following properties:

- 688 • state: a URI that represents a potential state.

689 5.3 Example

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

```
697 <RelationshipTypes>
698
699   <RelationshipType id="processDeployedOn"
700     name="Process is deployed on"
701     semantics="www.my.com/RelSemantics/procDeployedOn"
702     cascadingDeletion="yes">
703
704     <RelationshipTypeProperties element="ProcessDeployedOnProperties"/>
705
706     <InstanceStates>
707       <InstanceState state="www.my.com/successfullyDeployed"/>
708       <InstanceState state="www.my.com/failed"/>
709     </InstanceStates>
710
711   </RelationshipType>
712
713 </RelationshipTypes>
```

714 6 Topology Template

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

720 6.1 Syntax

```
721 1   <TopologyTemplate id="ID"
722 2     name="string"?>
723 3
724 4   (
725 5     <NodeTemplate id="ID"
726 6       name="string"?
727 7       nodeType="QName"
728 8       minInstances="int"?
729 9       maxInstances="int|string"?>
730 10
731 11     <PropertyDefaults?
732 12       XML fragment
733 13     </PropertyDefaults>
```

```

734 14
735 15     <PropertyConstraints>?
736 16
737 17         <PropertyConstraint property="string"
738 18             constraintType="anyURI">+
739 19             constraint?
740 20         </PropertyConstraint>
741 21
742 22     </PropertyConstraints>
743 23
744 24     <Policies>?
745 25         <Policy name="string" type="anyURI">+
746 26             policy specific content
747 27         </Policy>
748 28     </Policies>
749 29
750 30     <EnvironmentConstraints>?
751 31         <EnvironmentConstraint constraintType="anyURI">+
752 32             constraint type specific content?
753 33         </EnvironmentConstraint>
754 34     </EnvironmentConstraints>
755 35
756 36     <DeploymentArtifacts>?
757 37         <DeploymentArtifact name="string" type="anyURI">+
758 38             artifact specific content
759 39         </DeploymentArtifact>
760 40     </DeploymentArtifacts>
761 41
762 42 </NodeTemplate>
763 43 |
764 44 <RelationshipTemplate id="ID"
765 45     name="string"?
766 46     relationshipType="QName">
767 47
768 48     <SourceElement id="IDREF"/>
769 49
770 50     ( <TargetElement id="IDREF"/>
771 51     |
772 52     <TargetElementReference id="QName"/>
773 53     )
774 54
775 55     <PropertyDefaults>?
776 56         XML fragment
777 57     </PropertyDefaults>
778 58
779 59     <PropertyConstraints>?
780 60
781 61         <PropertyConstraint property="string"
782 62             constraintType="anyURI">+
783 63             constraint?
784 64         </PropertyConstraint>
785 65
786 66     </PropertyConstraints>
787 67
788 68     <RelationshipConstraints>?
789 69
790 70         <RelationshipConstraint constraintType="anyURI">+
791 71             constraint?

```

```

792 72         </RelationshipConstraint>
793 73     </RelationshipConstraints>
794 74 </RelationshipTemplate>
795 75 |
796 76 <GroupTemplate id="ID"
797 77     name="string"?
798 78     minInstances="int"?
799 79     maxInstances="int|string"?>
800 80     (
801 81         <NodeTemplate ... />
802 82         |
803 83         <RelationshipTemplate ... />
804 84         |
805 85         <GroupTemplate ... />
806 86     )+
807 87 <Policies>?
808 88     <Policy name="string" type="anyURI">+
809 89         policy specific content
810 90     </Policy>
811 91 </Policies>
812 92 </GroupTemplate>
813 93 )+
814 94 </TopologyTemplate>
815 95
816 96
817 97
818 98
819 99
820 100

```

821 6.2 Properties

822 The `TopologyTemplate` element has the following properties:

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

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

837 The `NodeTemplate` element has the following properties:

- 838 • `id`: This attribute specifies the identifier of the Node Template. The identifier of the Node
839 Template MUST be unique within the target namespace.
- 840 • `name`: This optional attribute specifies the name of the Node Template.

- 841 • `nodeType`: The QName value of this attribute refers to the Node Type providing the type of the
842 Node Template.
- 843 • `minInstances`: This integer attribute specifies the minimum number of instances to be created
844 when instantiating the Node Template. The default value of this attribute is 1..The value of
845 `minInstances` MUST NOT be less than 0.
- 846 • `maxInstances`: This attribute specifies the maximum number of instances that can be created
847 when instantiating the Node Template. The default value of this attribute is 1. If the string is set to
848 “unbounded”, an unbounded number of instances can be created. The value of `maxInstances`
849 MUST be 1 or greater and MUST NOT be less than the value specified for `minInstances`.
- 850 • `PropertyDefaults`: Specifies initial values for one or more of the Node Type Properties of the
851 Node Type providing the property definitions in the concrete context of the Node Template.
- 852 The initial values are specified by providing an instance document of the XML schema of the
853 corresponding Node Type Properties. This instance document considers the inheritance structure
854 deduced by the `DerivedFrom` property of the Node Type referenced by the `nodeType` attribute
855 of the Node Template.
- 856 The instance document of the XML schema might not validate against the existence constraints
857 of the corresponding schema: not all node type properties might have an initial value assigned,
858 i.e. mandatory elements or attributes might be missing in the instance provided by the Property
859 Defaults element. Once the defined Node Template has been instantiated, any XML
860 representation of the Node Type properties MUST validate according to the associated XML
861 schema definition.
- 862 • `PropertyConstraints`: Specifies constraints on the use of one or more of the Node Type
863 Properties of the Node Type providing the property definitions for the Node Template.
- 864 Each constraint is specified by means of a separate nested `PropertyConstraint` element. This
865 element contains the actual encoding of the constraint.
- 866 • `Policies`: Specifies policies of the Node Template. Each policy is specified by means of a
867 separate nested `Policy` element. This element contains the actual policy specific content of the
868 policy.
- 869 Note, that a policy specified in the Node Template overrides any policy of the same name and
870 type that might be specified with the Node Type of this Node Template.
- 871 Any policies of the Node Type that are not overridden are combined with the policies of the Node
872 Template.
- 873 • `EnvironmentConstraints`: The nested `EnvironmentConstraint` elements of the Node
874 Template under definition constrain the runtime environment for the corresponding component of
875 a service. For example, constraints on network security settings of the hosting environment or
876 requirements on the existence of certain resources might be defined within the Environment
877 Constraints definition of a Node Template.
- 878 • `DeploymentArtifacts`: This element specifies the deployment artifacts relevant for the Node
879 Template under definition.
- 880 Its nested `DeploymentArtifact` elements specify details about individual deployment artifacts.
881 The `name` attribute of a `DeploymentArtifact` element specifies the name of the artifact.
882 Uniqueness of the name within the scope of the encompassing Node Template SHOULD be
883 guaranteed by the definition. The `type` attribute of a `DeploymentArtifact` element specifies
884 the type of the deployment artifact definition that is related to the Node Template, i.e. the attribute
885 gives a hint how to interpret the body of the `DeploymentArtifact` element. The body of this
886 element contains the type-specific content.

890
891 For example, if the `type` attribute contains the value
892 `http://docs.oasis-open.org/tosca/ns/2011/12/deploymentArtifacts/ovfRef`, the body will contain an
893 XML fragment with a reference to an OVF package and a mapping between service template
894 data and elements of the respective OVF envelope.

895
896 Note, that a Deployment Artifact specified with the Node Template under definition overrides any
897 Deployment Artifact of the same name and the same type specified with the Node Type given as
898 value of the `nodeType` attribute of the Node Template under definition.

899
900 Otherwise, the Deployment Artifacts of the Node Type given as value of the `nodeType` attribute
901 of the Node Template under definition and the Deployment Artifacts defined with the Node
902 Template are combined.

903

904 The `PropertyConstraint` element has the following properties:

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

910

911 For example, a constraint type of `http://www.example.com/PropertyConstraints/unique` could
912 denote that the reference property of the node template under definition has to be unique within a
913 certain scope. The constraint type specific content of the respective `PropertyConstraint`
914 element could then define the actual scope in which uniqueness has to be ensured in more detail.

915 The `Policy` element has the following properties:

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

920 The `EnvironmentConstraint` element has the following properties:

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

923 The `RelationshipTemplate` element has the following properties:

- 924 • `id`: This attribute specifies the identifier of the Relationship Template. The identifier of the
925 Relationship Template MUST be unique within the target namespace.
- 926 • `name`: This optional attribute specifies the name of the Relationship Template.
- 927 • `relationshipType`: The QName value of this property refers to the Relationship Type
928 providing the type of the Relationship Template.
- 929 • `SourceElement`: The `id` attribute of this element references a Node Template or Group
930 Template within the same Service Template document that is the source of the Relationship
931 Template.
- 932 • `TargetElement`: The `id` attribute of this element references a Node Template or Group
933 Template within the same Service Template document that is the target of the Relationship
934 Template.

935 • `TargetElementReference`: The `id` attribute of this element refers by QName to an imported
936 Node Template or Group Template that is the target of the Relationship Template. The
937 referenced Node Template or Group Template will typically be the root node or root group of the
938 corresponding Topology Template. In some cases a non-root Node Template or non-root Group
939 Template might be referenced to support access to particular resources from a larger service, for
940 example. Either `TargetElement` or `TargetElementReference` MUST be specified but not
941 both.

942 • `PropertyDefaults`: Specifies initial values for one or more of the Relationship Type Properties
943 of the Relationship Type providing the property definitions in the concrete context of the
944 Relationship Template.

945 The initial values are specified by providing an instance document of the XML schema of the
946 corresponding Relationship Type Properties.

947 The instance document of the XML schema might not validate against the existence constraints
948 of the corresponding schema: not all relationship type properties might have an initial value
949 assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the
950 Property Defaults element. Once the defined Relationship Template has been instantiated, any
951 XML representation of the Relationship Type properties MUST validate according to the
952 associated XML schema definition.

953 • `PropertyConstraints`: Specifies constraints on the use of one or more of the Relationship
954 Type Properties of the Relationship Type providing the property definitions for the Relationship
955 Template.

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

958 • `RelationshipConstraints`: Specifies constraints on the use of the relationship.

959 Each constraint is specified by means of a separate nested `RelationshipConstraint`
960 element. This element can contain the actual encoding of the constraint, or its `constraintType`
961 attribute already denotes the constraint itself. The constraint type is specified by means of a URI,
962 which defines both the semantic meaning of the constraint as well as the format of any content.

963 The `GroupTemplate` element has the following properties:

964 • `id`: This attribute specifies the identifier of the Group Template. The identifier of the Group
965 Template MUST be unique within the target namespace.

966 • `name`: This optional attribute specifies the name of the Group Template.

967 • `minInstances`: This integer attribute specifies the minimum number of instances to be created
968 when instantiating the Group Template. The default value of this attribute is 1. The value of
969 `minInstances` MUST NOT be less than 0.

970 • `maxInstances`: This attribute specifies the maximum number of instances that can be created
971 when instantiating the Group Template. The default value of this attribute is 1. If the string is set
972 to "unbounded", an unbounded number of instances can be created. The value of `maxInstances`
973 MUST be 1 or greater and MUST NOT be less than the value specified for `minInstances`.

974 • `NodeTemplate`: This is a node template contained within, or grouped by the Group Template.

975 • `RelationshipTemplate`: This is a relationship template contained within, or grouped by the
976 Group.

977 • `GroupTemplate`: This is a Group Template of a nested group contained within, or grouped by
978 the Group Template.

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

982 6.3 Example

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

```
995 <ServiceTemplate id="myService"  
996     name="My Service"  
997     targetNamespace="http://www.ibm.com/sample"  
998     xmlns:abc="http://www.ibm.com/sample">  
999  
1000     <Import namespace="http://www.ibm.com/sample"  
1001         importType="http://docs.oasis-open.org/tosca/ns/2011/12"/>  
1002  
1003     <TopologyTemplate id="SampleApplication">  
1004  
1005         <NodeTemplate id="MyApplication"  
1006             name="My Application"  
1007             nodeType="abc:Application">  
1008             <PropertyDefaults>  
1009                 <ApplicationProperties>  
1010                     <Owner>Frank</Owner>  
1011                     <InstanceName>Thomas' favorite application</InstanceName>  
1012                 </ApplicationProperties>  
1013             </PropertyDefaults>  
1014         </NodeTemplate/>  
1015  
1016         <NodeTemplate id="MyAppServer"  
1017             name="My Application Server"  
1018             nodeType="abc:ApplicationServer"  
1019             minInstances="0"  
1020             maxInstances="unbounded"/>  
1021  
1022         <RelationshipTemplate id="MyDeploymentRelationship"  
1023             relationshipType="deployedOn">  
1024             <SourceElement id="MyApplication"/>  
1025             <TargetElement id="MyAppServer"/>  
1026         </RelationshipTemplate>  
1027     </TopologyTemplate>  
1028 </ServiceTemplate>
```

1031

7 Plans

1032 The operational management behavior of a Service Template is invoked by means of orchestration plans,
1033 or more simply, *Plans*. Plans consist of individual steps (aka tasks or activities) to be performed and the
1034 definition of the potential order of these steps. The execution of a step can be performed by one of the
1035 functions offered via the interfaces of a Node Template, by invoking operations of a Service Template
1036 API, or by invoking other operations being required in the context of a specific service. Plans are
1037 classified by a type, and the following two plan types are defined as part of the TOSCA specification.
1038 *Build plans* specify how instances of their associated Service Templates are made, and *termination plans*
1039 specify how an instance of a Service Template is removed from the environment. Other plan types for
1040 managing existing service instances throughout their life time are termed *modification plans*, and it is
1041 expected that such plan types will be defined subsequently by authors of service templates and domain
1042 expert groups.

1043

7.1 Syntax

1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064

```
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>
```

1065

7.2 Properties

1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077

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

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

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

- `languageUsed`: This attribute denotes the process modeling language (or metamodel) used to specify the plan. For example, "<http://www.omg.org/spec/BPMN/2.0/>" would specify that BPMN 2.0 has been used to model the plan.
- `PreCondition`: This optional element specifies a condition that needs to be satisfied in order for the plan to be executed. The `expressionLanguage` attribute of this element specifies the expression language the nested condition is provided in.
Typically, the precondition will be an expression in the instance state attribute of some of the node templates or relationship templates of the topology template. It will be evaluated based on the actual values of the corresponding attributes at the time the plan is requested to be executed. Note, that any other kind of pre-condition is allowed.
- `PlanModel`: This property contains the actual model content.
- `PlanModelReference`: This property points to the model content. Its reference attribute contains a URI of the model of the plan.

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

7.3 Use of Process Modeling Languages

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

7.4 Example

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

```
<Plans>
  <Plan id="DeployApplication"
    name="Sample Application Build Plan"
    planType=
      "http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
    languageUsed="http://www.omg.org/spec/BPMN/2.0/">
    <PreCondition expressionLanguage="www.my.com/text">?
      Run only if funding is available
    </PreCondition>
  <PlanModel>
    <process name="DeployNewApplication" id="p1">
```

```

1126     <documentation>This process deploys a new instance of the
1127         sample application.
1128     </documentation>
1129
1130     <task id="t1" name="CreateAccount"/>
1131
1132     <task id="t2" name="AcquireNetworkAddresses"
1133         isSequential="false"
1134         loopDataInput="t2Input.LoopCounter"/>
1135         <documentation>Assumption: t2 gets data of type "input"
1136             as input and this data has a field names "LoopCounter"
1137             that contains the actual multiplicity of the task.
1138         </documentation>
1139
1140     <task id="t3" name="DeployApplicationServer"
1141         isSequential="false"
1142         loopDataInput="t3Input.LoopCounter"/>
1143
1144     <task id="t4" name="DeployApplication"
1145         isSequential="false"
1146         loopDataInput="t4Input.LoopCounter"/>
1147
1148     <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
1149     <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
1150     <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
1151 </process>
1152 </PlanModel>
1153 </Plan>
1154
1155 <Plan id="RemoveApplication"
1156     planType="http://docs.oasis-
1157         open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
1158     languageUsed=
1159         "http://docs.oasis-open.org/wsbpel/2.0/process/executable">
1160     <PlanModelReference reference="prj:RemoveApp"/>
1161 </Plan>
1162
1163 </Plans>

```

1164 8 Security Considerations

1165 TOSCA does not mandate the use of any specific mechanism or technology for client authentication.
1166 However, a client MUST provide a principal or the principal MUST be obtainable by the infrastructure.

1167 **9 Conformance**

1168 **This section is to be done.**

1169 **Appendix A. Portability and Interoperability**
1170 **Considerations**

1171 This section illustrates the portability and interoperability aspects addressed by Service Templates:
1172 Portability - The ability to take Service Templates created in one vendor's environment and use them in
1173 another vendor's environment.
1174 Interoperability - The capability for multiple components (e.g. a task of a plan and the definition of a
1175 topology node) to interact using well-defined messages and protocols. This enables combining
1176 components from different vendors allowing seamless management of services.
1177 Portability demands support of TOSCA artifacts.

Appendix B. Complete TOSCA Grammar

```

1179 1 <ServiceTemplate id="ID"
1180 2     name="string"?
1181 3     targetNamespace="anyURI">
1182 4
1183 5 <Extensions>?
1184 6 <Extension namespace="anyURI"
1185 7     mustUnderstand="yes|no"?/>+
1186 8 </Extensions>
1187 9
1188 10 <Import namespace="anyURI"?
1189 11     location="anyURI"?
1190 12     importType="anyURI"/>*
1191 13
1192 14 <Types>?
1193 15 <xs:schema .../>*
1194 16 </Types>
1195 17
1196 18 (
1197 19 <TopologyTemplateReference reference="QName"/>
1198 20 |
1199 21 <TopologyTemplate id="ID"
1200 22     name="string"?>
1201 23
1202 24 (
1203 25 <NodeTemplate id="ID"
1204 26     name="string"?
1205 27     nodeType="QName"
1206 28     minInstances="int"?
1207 29     maxInstances="int|string"?>
1208 30
1209 31 <PropertyDefaults>?
1210 32     XML fragment
1211 33 </PropertyDefaults>
1212 34
1213 35 <PropertyConstraints>?
1214 36
1215 37 <PropertyConstraint property="string"
1216 38     constraintType="anyURI">+
1217 39     constraint?
1218 40 </PropertyConstraint>
1219 41
1220 42 </PropertyConstraints>
1221 43
1222 44 <Policies>?
1223 45 <Policy name="string" type="anyURI">+
1224 46     policy specific content
1225 47 </Policy>
1226 48 </Policies>
1227 49
1228 50 <EnvironmentConstraints>?
1229 51 <EnvironmentConstraint constraintType="anyURI">+
1230 52     constraint type specific content?
1231 53 </EnvironmentConstraint>

```



```

1232 54     </EnvironmentConstraints>
1233 55
1234 56     <DeploymentArtifacts>?
1235 57         <DeploymentArtifact name="string" type="anyURI">+
1236 58             artifact specific content
1237 59         </DeploymentArtifact>
1238 60     </DeploymentArtifacts>
1239 61
1240 62 </NodeTemplate>
1241 63 |
1242 64 <RelationshipTemplate id="ID"
1243 65     name="string"?
1244 66     relationshipType="QName">+
1245 67
1246 68     <SourceElement id="IDREF"/>
1247 69
1248 70     ( <TargetElement id="IDREF"/>
1249 71         |
1250 72         <TargetElementReference id="QName"/>
1251 73     )
1252 74
1253 75     <PropertyDefaults>?
1254 76         XML fragment
1255 77     </PropertyDefaults>
1256 78
1257 79     <PropertyConstraints>?
1258 80
1259 81         <PropertyConstraint property="string"
1260 82             constraintType="anyURI">+
1261 83             constraint?
1262 84         </PropertyConstraint>
1263 85
1264 86     </PropertyConstraints>
1265 87
1266 88     <RelationshipConstraints>?
1267 89
1268 90         <RelationshipConstraint constraintType="anyURI">+
1269 91             constraint?
1270 92         </RelationshipConstraint>
1271 93
1272 94     </RelationshipConstraints>
1273 95
1274 96 </RelationshipTemplate>
1275 97 |
1276 98 <GroupTemplate id="ID"
1277 99     name="string"?
1278 100     minInstances="int"?
1279 101     maxInstances="int|string"?>
1280 102
1281 103     (
1282 104         <NodeTemplate ... />
1283 105         |
1284 106         <RelationshipTemplate ... />
1285 107         |
1286 108         <GroupTemplate ... />
1287 109     )+
1288 110
1289 111 <Policies>?

```

```

1290 112         <Policy name="string" type="anyURI">+
1291 113             policy specific content
1292 114         </Policy>
1293 115     </Policies>
1294 116
1295 117     </GroupTemplate>
1296 118 )+
1297 119
1298 120 </TopologyTemplate>
1299 121 )?
1300 122
1301 123 <NodeTypes>?
1302 124
1303 125     <NodeType id="ID"
1304 126         name="string"?>+
1305 127
1306 128     <NodeTypeProperties element="QName"?
1307 129         type="QName"?/>?
1308 130
1309 131     <DerivedFrom nodeTypeRef="QName"/>?
1310 132
1311 133     <InstanceStates>?
1312 134         <InstanceState state="anyURI">+
1313 135     </InstanceStates>
1314 136
1315 137     <Interfaces>?
1316 138
1317 139     <Interface>+
1318 140
1319 141         (
1320 142         <WSDL portType="QName"
1321 143             operation="NCName"?>+
1322 144         |
1323 145         <REST method="GET | PUT | POST | DELETE"
1324 146             requestURI="anyURI"
1325 147             requestPayload="QName"?
1326 148             responsePayload="QName"?>+
1327 149         |
1328 150         <Operation name="NCame">+
1329 151
1330 152         <InputParameters>?
1331 153
1332 154             <InputParamter name="string"
1333 155                 type="string"
1334 156                 required="yes|no">+
1335 157
1336 158         </InputParameters>
1337 159
1338 160         <OutputParameters>?
1339 161
1340 162             <OutputParamter name="string"
1341 163                 type="string"
1342 164                 required="yes|no">+
1343 165
1344 166         </OutputParameters>
1345 167
1346 168         <Implementations>
1347 169

```

```

1348 170         <Implementation implementationID="anyURI"?
1349 171             language="anyURI"?>+
1350 172         (
1351 173             <ImplementationProper>?
1352 174                 code
1353 175             </ImplementationProper>
1354 176             |
1355 177             <ImplementationReference ref="anyURI"/>?
1356 178         )
1357 179         <Implementation>
1358 180
1359 181     </Implementations>
1360 182 </Operation>
1361 183 )
1362 184
1363 185 </Interface>
1364 186
1365 187 </Interfaces>
1366 188
1367 189 <DeploymentArtifacts>?
1368 190     <DeploymentArtifact name="string" type="anyURI">+
1369 191         artifact specific content
1370 192     </DeploymentArtifact>
1371 193 </DeploymentArtifacts>
1372 194
1373 195
1374 196 <Policies>?
1375 197
1376 198     <Policy name="string" type="anyURI">+
1377 199         policy specific content
1378 200     </Policy>
1379 201
1380 202 </Policies>
1381 203
1382 204 </NodeType>
1383 205
1384 206 </NodeTypes>
1385 207
1386 208 <RelationshipTypes>?
1387 209
1388 210     <RelationshipType id="ID"
1389 211         name="string"?
1390 212         semantics="anyURI"
1391 213         cascadingDeletion="yes|no"?>+
1392 214
1393 215     <RelationshipTypeProperties element="QName"?
1394 216         type="QName"?/>?
1395 217
1396 218     <InstanceStates>?
1397 219         <InstanceState state="anyURI">+
1398 220     </InstanceStates>
1399 221
1400 222     </RelationshipType>
1401 223
1402 224 </RelationshipTypes>
1403 225
1404 226 <Plans>?
1405 227

```

```
1406 228     <Plan id="ID"
1407 229         name="string"?
1408 230         planType="anyURI"
1409 231         languageUsed="anyURI">+
1410 232
1411 233     <PreCondition expressionLanguage="anyURI">?
1412 234         condition
1413 235     </PreCondition>
1414 236
1415 237     ( <PlanModel>
1416 238         actual plan
1417 239     </PlanModel>
1418 240     |
1419 241     <PlanModelReference reference="anyURI"/>
1420 242     )
1421 243
1422 244     </Plan>
1423 245
1424 246     </Plans>
1425 247
1426 248 </ServiceTemplate>
```

Appendix C. TOSCA Schema

```
1428 1 <?xml version="1.0" encoding="UTF-8"?>
1429 2 <xs:schema targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
1430 3   elementFormDefault="qualified" attributeFormDefault="unqualified"
1431 4   xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
1432 5   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1433 6
1434 7   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
1435 8     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
1436 9
1437 10  <xs:element name="documentation" type="tDocumentation"/>
1438 11  <xs:complexType name="tDocumentation" mixed="true">
1439 12    <xs:sequence>
1440 13      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1441 14    </xs:sequence>
1442 15    <xs:attribute name="source" type="xs:anyURI"/>
1443 16    <xs:attribute ref="xml:lang"/>
1444 17  </xs:complexType>
1445 18
1446 19  <xs:complexType name="tExtensibleElements">
1447 20    <xs:sequence>
1448 21      <xs:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
1449 22      <xs:any namespace="##other" processContents="lax" minOccurs="0"
1450 23        maxOccurs="unbounded"/>
1451 24    </xs:sequence>
1452 25    <xs:anyAttribute namespace="##other" processContents="lax"/>
1453 26  </xs:complexType>
1454 27
1455 28  <xs:complexType name="tImport">
1456 29    <xs:complexContent>
1457 30      <xs:extension base="tExtensibleElements">
1458 31        <xs:attribute name="namespace" type="xs:anyURI"/>
1459 32        <xs:attribute name="location" type="xs:anyURI"/>
1460 33        <xs:attribute name="importType" type="importedURI" use="required"/>
1461 34      </xs:extension>
1462 35    </xs:complexContent>
1463 36  </xs:complexType>
1464 37
1465 38  <xs:element name="ServiceTemplate">
1466 39    <xs:complexType>
1467 40      <xs:complexContent>
1468 41        <xs:extension base="tServiceTemplate"/>
1469 42      </xs:complexContent>
1470 43    </xs:complexType>
1471 44  </xs:element>
1472 45
1473 46  <xs:complexType name="tServiceTemplate">
1474 47    <xs:complexContent>
1475 48      <xs:extension base="tExtensibleElements">
1476 49        <xs:sequence>
1477 50          <xs:element name="Import" type="tImport" minOccurs="0"
1478 51            maxOccurs="unbounded"/>
1479 52          <xs:element name="Types" minOccurs="0">
1480 53            <xs:complexType>
```

```

1481 54     <xs:sequence>
1482 55     <xs:any namespace="##other" processContents="lax" minOccurs="0"
1483 56         maxOccurs="unbounded"/>
1484 57     </xs:sequence>
1485 58     </xs:complexType>
1486 59 </xs:element>
1487 60 <xs:element name="Extensions" minOccurs="0">
1488 61     <xs:complexType>
1489 62         <xs:sequence>
1490 63             <xs:element name="Extension" type="tExtension"
1491 64                 maxOccurs="unbounded"/>
1492 65         </xs:sequence>
1493 66     </xs:complexType>
1494 67 </xs:element>
1495 68 <xs:choice minOccurs="0">
1496 69     <xs:element name="TopologyTemplateReference">
1497 70         <xs:complexType>
1498 71             <xs:attribute name="reference" type="xs:QName"/>
1499 72         </xs:complexType>
1500 73     </xs:element>
1501 74     <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1502 75 </xs:choice>
1503 76 <xs:element name="NodeTypes" type="tNodeTypes" minOccurs="0"/>
1504 77 <xs:element name="RelationshipTypes" type="tRelationshipTypes"
1505 78     minOccurs="0"/>
1506 79 <xs:element name="Plans" type="tPlans" minOccurs="0"/>
1507 80 </xs:sequence>
1508 81 <xs:attribute name="id" type="xs:ID" use="required"/>
1509 82 <xs:attribute name="name" type="xs:string" use="optional"/>
1510 83 <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1511 84 </xs:extension>
1512 85 </xs:complexContent>
1513 86 </xs:complexType>
1514 87
1515 88 <xs:complexType name="tDeploymentArtifact">
1516 89     <xs:complexContent>
1517 90         <xs:extension base="tExtensibleElements">
1518 91             <xs:attribute name="name" type="xs:string" use="required"/>
1519 92             <xs:attribute name="type" type="xs:anyURI" use="required"/>
1520 93         </xs:extension>
1521 94     </xs:complexContent>
1522 95 </xs:complexType>
1523 96
1524 97 <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1525 98 <xs:complexType name="tNodeTemplate">
1526 99     <xs:complexContent>
1527 100         <xs:extension base="tExtensibleElements">
1528 101             <xs:sequence>
1529 102                 <xs:element name="PropertyDefaults" minOccurs="0">
1530 103                     <xs:complexType>
1531 104                         <xs:sequence>
1532 105                             <xs:any namespace="##other" processContents="lax"/>
1533 106                         </xs:sequence>
1534 107                     </xs:complexType>
1535 108                 </xs:element>
1536 109                 <xs:element name="PropertyConstraints" minOccurs="0">
1537 110                     <xs:complexType>
1538 111                         <xs:sequence>

```

```

1539 112     <xs:element name="PropertyConstraint"
1540 113         type="tPropertyConstraint" maxOccurs="unbounded"/>
1541 114     </xs:sequence>
1542 115     </xs:complexType>
1543 116 </xs:element>
1544 117 <xs:element name="Policies" minOccurs="0">
1545 118     <xs:complexType>
1546 119         <xs:sequence>
1547 120             <xs:element name="Policy" type="tPolicy"
1548 121                 maxOccurs="unbounded"/>
1549 122         </xs:sequence>
1550 123     </xs:complexType>
1551 124 </xs:element>
1552 125 <xs:element name="DeploymentArtifacts" minOccurs="0">
1553 126     <xs:complexType>
1554 127         <xs:sequence>
1555 128             <xs:element name="DeploymentArtifact"
1556 129                 type="tDeploymentArtifact" maxOccurs="unbounded"/>
1557 130         </xs:sequence>
1558 131     </xs:complexType>
1559 132 </xs:element>
1560 133 <xs:element name="EnvironmentConstraints" minOccurs="0">
1561 134     <xs:complexType>
1562 135         <xs:sequence>
1563 136             <xs:element name="EnvironmentConstraint"
1564 137                 type="tEnvironmentConstraint" maxOccurs="unbounded"/>
1565 138         </xs:sequence>
1566 139     </xs:complexType>
1567 140 </xs:element>
1568 141 </xs:sequence>
1569 142 <xs:attribute name="id" type="xs:ID" use="required"/>
1570 143 <xs:attribute name="name" type="xs:string" use="optional"/>
1571 144 <xs:attribute name="nodeType" type="xs:QName" use="required"/>
1572 145 <xs:attribute name="minInstances" type="xs:int" use="optional"
1573 146     default="1"/>
1574 147 <xs:attribute name="maxInstances" use="optional" default="1">
1575 148     <xs:simpleType>
1576 149         <xs:union>
1577 150             <xs:simpleType>
1578 151                 <xs:restriction base="xs:nonNegativeInteger">
1579 152                     <xs:pattern value="([1-9]+[0-9]*)"/>
1580 153                 </xs:restriction>
1581 154             </xs:simpleType>
1582 155             <xs:simpleType>
1583 156                 <xs:restriction base="xs:string">
1584 157                     <xs:enumeration value="unbounded"/>
1585 158                 </xs:restriction>
1586 159             </xs:simpleType>
1587 160         </xs:union>
1588 161     </xs:simpleType>
1589 162 </xs:attribute>
1590 163 </xs:extension>
1591 164 </xs:complexContent>
1592 165 </xs:complexType>
1593 166
1594 167 <xs:complexType name="tPropertyConstraint">
1595 168     <xs:sequence>
1596 169 <xs:any namespace="##other" processContents="lax" minOccurs="0"/>

```

```

1597 170     </xs:sequence>
1598 171     <xs:attribute name="property" type="xs:string" use="required"/>
1599 172     <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
1600 173 </xs:complexType>
1601 174
1602 175 <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
1603 176 <xs:complexType name="tTopologyTemplate">
1604 177   <xs:complexContent>
1605 178     <xs:extension base="tTopologyElementCollection"/>
1606 179   </xs:complexContent>
1607 180 </xs:complexType>
1608 181
1609 182 <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1610 183 <xs:complexType name="tGroupTemplate">
1611 184   <xs:complexContent>
1612 185     <xs:extension base="tTopologyElementCollection">
1613 186       <xs:sequence>
1614 187         <xs:element name="Policies" minOccurs="0">
1615 188           <xs:complexType>
1616 189             <xs:sequence>
1617 190               <xs:element name="Policy" type="tPolicy"
1618 191                 maxOccurs="unbounded"/>
1619 192             </xs:sequence>
1620 193           </xs:complexType>
1621 194         </xs:element>
1622 195       </xs:sequence>
1623 196       <xs:attribute name="minInstances" type="xs:int" use="optional"
1624 197         default="1"/>
1625 198       <xs:attribute name="maxInstances" use="optional" default="1">
1626 199         <xs:simpleType>
1627 200           <xs:union>
1628 201             <xs:simpleType>
1629 202               <xs:restriction base="xs:nonNegativeInteger">
1630 203                 <xs:pattern value="([1-9]+[0-9]*)"/>
1631 204               </xs:restriction>
1632 205             </xs:simpleType>
1633 206             <xs:simpleType>
1634 207               <xs:restriction base="xs:string">
1635 208                 <xs:enumeration value="unbounded"/>
1636 209               </xs:restriction>
1637 210             </xs:simpleType>
1638 211           </xs:union>
1639 212         </xs:simpleType>
1640 213       </xs:attribute>
1641 214     </xs:extension>
1642 215   </xs:complexContent>
1643 216 </xs:complexType>
1644 217
1645 218 <xs:complexType name="tTopologyElementCollection">
1646 219   <xs:complexContent>
1647 220     <xs:extension base="tExtensibleElements">
1648 221       <xs:choice maxOccurs="unbounded">
1649 222         <xs:element name="NodeTemplate" type="tNodeTemplate"/>
1650 223         <xs:element name="RelationshipTemplate"
1651 224           type="tRelationshipTemplate"/>
1652 225         <xs:element name="GroupTemplate" type="tGroupTemplate"/>
1653 226       </xs:choice>
1654 227     <xs:attribute name="id" type="xs:ID" use="required"/>

```



```

1655 228     <xs:attribute name="name" type="xs:string" use="optional"/>
1656 229     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1657 230     </xs:extension>
1658 231     </xs:complexContent>
1659 232 </xs:complexType>
1660 233
1661 234 <xs:element name="RelationshipTypes" type="tRelationshipTypes"/>
1662 235 <xs:complexType name="tRelationshipTypes">
1663 236   <xs:sequence>
1664 237     <xs:element name="RelationshipType" type="tRelationshipType"
1665 238       maxOccurs="unbounded"/>
1666 239   </xs:sequence>
1667 240   <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1668 241 </xs:complexType>
1669 242
1670 243 <xs:element name="RelationshipType" type="tRelationshipType"/>
1671 244 <xs:complexType name="tRelationshipType">
1672 245   <xs:complexContent>
1673 246     <xs:extension base="tExtensibleElements">
1674 247       <xs:sequence>
1675 248         <xs:element name="RelationshipTypeProperties" minOccurs="0">
1676 249           <xs:complexType>
1677 250             <xs:attribute name="element" type="xs:QName"/>
1678 251             <xs:attribute name="type" type="xs:QName"/>
1679 252           </xs:complexType>
1680 253         </xs:element>
1681 254         <xs:element name="InstanceStates"
1682 255           type="tTopologyElementInstanceStates" minOccurs="0"/>
1683 256       </xs:sequence>
1684 257       <xs:attribute name="id" type="xs:ID" use="required"/>
1685 258       <xs:attribute name="name" type="xs:string" use="optional"/>
1686 259       <xs:attribute name="semantics" type="xs:anyURI" use="required"/>
1687 260       <xs:attribute name="cascadingDeletion" type="tBoolean"
1688 261         use="optional" default="no"/>
1689 262       <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1690 263     </xs:extension>
1691 264   </xs:complexContent>
1692 265 </xs:complexType>
1693 266
1694 267 <xs:element name="RelationshipTemplate" type="tRelationshipTemplate"/>
1695 268 <xs:complexType name="tRelationshipTemplate">
1696 269   <xs:complexContent>
1697 270     <xs:extension base="tExtensibleElements">
1698 271       <xs:sequence>
1699 272         <xs:element name="SourceElement">
1700 273           <xs:complexType>
1701 274             <xs:attribute name="id" type="xs:IDREF" use="required"/>
1702 275           </xs:complexType>
1703 276         </xs:element>
1704 277         <xs:choice>
1705 278           <xs:element name="TargetElement">
1706 279             <xs:complexType>
1707 280               <xs:attribute name="id" type="xs:IDREF" use="required"/>
1708 281             </xs:complexType>
1709 282           </xs:element>
1710 283           <xs:element name="TargetElementReference">
1711 284             <xs:complexType>
1712 285               <xs:attribute name="id" type="xs:QName" use="required"/>

```

```

1713 286     </xs:complexType>
1714 287     </xs:element>
1715 288 </xs:choice>
1716 289 <xs:element name="PropertyDefaults" minOccurs="0">
1717 290   <xs:complexType>
1718 291     <xs:sequence>
1719 292       <xs:any namespace="##other" processContents="lax"/>
1720 293     </xs:sequence>
1721 294   </xs:complexType>
1722 295 </xs:element>
1723 296 <xs:element name="PropertyConstraints" minOccurs="0">
1724 297   <xs:complexType>
1725 298     <xs:sequence>
1726 299       <xs:element name="PropertyConstraint"
1727 300         type="tPropertyConstraint" maxOccurs="unbounded"/>
1728 301     </xs:sequence>
1729 302   </xs:complexType>
1730 303 </xs:element>
1731 304 <xs:element name="RelationshipConstraints" minOccurs="0">
1732 305   <xs:complexType>
1733 306     <xs:sequence>
1734 307       <xs:element name="RelationshipConstraint"
1735 308         maxOccurs="unbounded">
1736 309         <xs:complexType>
1737 310           <xs:sequence>
1738 311             <xs:any namespace="##other" processContents="lax"
1739 312               minOccurs="0"/>
1740 313           </xs:sequence>
1741 314           <xs:attribute name="constraintType" type="xs:anyURI"
1742 315             use="required"/>
1743 316         </xs:complexType>
1744 317       </xs:element>
1745 318     </xs:sequence>
1746 319   </xs:complexType>
1747 320 </xs:element>
1748 321 </xs:sequence>
1749 322 <xs:attribute name="id" type="xs:ID" use="required"/>
1750 323 <xs:attribute name="name" type="xs:string" use="optional"/>
1751 324 <xs:attribute name="relationshipType" type="xs:QName"
1752 325   use="required"/>
1753 326 </xs:extension>
1754 327 </xs:complexContent>
1755 328 </xs:complexType>
1756 329
1757 330 <xs:element name="NodeTypes" type="tNodeTypes"/>
1758 331 <xs:complexType name="tNodeTypes">
1759 332   <xs:sequence>
1760 333     <xs:element name="NodeType" type="tNodeType" maxOccurs="unbounded"/>
1761 334   </xs:sequence>
1762 335   <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1763 336 </xs:complexType>
1764 337
1765 338 <xs:element name="NodeType" type="tNodeType"/>
1766 339 <xs:complexType name="tNodeType">
1767 340   <xs:complexContent>
1768 341     <xs:extension base="tExtensibleElements">
1769 342       <xs:sequence>
1770 343         <xs:element name="NodeTypeProperties" minOccurs="0">

```

```

1771 344     <xs:complexType>
1772 345         <xs:attribute name="element" type="xs:QName"/>
1773 346         <xs:attribute name="type" type="xs:QName"/>
1774 347     </xs:complexType>
1775 348 </xs:element>
1776 349 <xs:element name="DerivedFrom" minOccurs="0">
1777 350     <xs:complexType>
1778 351         <xs:attribute name="nodeTypeRef" type="xs:QName"
1779 352             use="required"/>
1780 353     </xs:complexType>
1781 354 </xs:element>
1782 355 <xs:element name="InstanceStates"
1783 356     type="tTopologyElementInstanceStates" minOccurs="0"/>
1784 357 <xs:element name="Interfaces" minOccurs="0">
1785 358     <xs:complexType>
1786 359         <xs:sequence>
1787 360             <xs:element name="Interface" maxOccurs="unbounded">
1788 361                 <xs:complexType>
1789 362                     <xs:choice>
1790 363                         <xs:element name="WSDL" type="tWSDL" maxOccurs="unbounded"/>
1791 364                         <xs:element name="REST" type="tREST" maxOccurs="unbounded"/>
1792 365                         <xs:element name="Operation" maxOccurs="unbounded">
1793 366                             <xs:complexType>
1794 367                                 <xs:complexContent>
1795 368                                     <xs:extension base="tOperation"/>
1796 369                                 </xs:complexContent>
1797 370                             </xs:complexType>
1798 371                         </xs:element>
1799 372                     </xs:choice>
1800 373                 </xs:complexType>
1801 374             </xs:element>
1802 375         </xs:sequence>
1803 376     </xs:complexType>
1804 377 </xs:element>
1805 378 <xs:element name="Policies" minOccurs="0">
1806 379     <xs:complexType>
1807 380         <xs:sequence>
1808 381             <xs:element name="Policy" type="tPolicy"
1809 382                 maxOccurs="unbounded"/>
1810 383         </xs:sequence>
1811 384     </xs:complexType>
1812 385 </xs:element>
1813 386 <xs:element name="DeploymentArtifacts" minOccurs="0">
1814 387     <xs:complexType>
1815 388         <xs:sequence>
1816 389             <xs:element name="DeploymentArtifact"
1817 390                 type="tDeploymentArtifact" maxOccurs="unbounded"/>
1818 391         </xs:sequence>
1819 392     </xs:complexType>
1820 393 </xs:element>
1821 394 </xs:sequence>
1822 395     <xs:attribute name="id" type="xs:ID" use="required"/>
1823 396     <xs:attribute name="name" type="xs:string" use="optional"/>
1824 397     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1825 398 </xs:extension>
1826 399 </xs:complexContent>
1827 400 </xs:complexType>
1828 401

```

```

1829 402 <xs:element name="Plans" type="tPlans"/>
1830 403 <xs:complexType name="tPlans">
1831 404 <xs:sequence>
1832 405 <xs:element name="Plan" type="tPlan" maxOccurs="unbounded"/>
1833 406 </xs:sequence>
1834 407 <xs:attribute name="targetNamespace" type="xs:anyURI"/>
1835 408 </xs:complexType>
1836 409
1837 410 <xs:element name="Plan" type="tPlan"/>
1838 411 <xs:complexType name="tPlan">
1839 412 <xs:complexContent>
1840 413 <xs:extension base="tExtensibleElements">
1841 414 <xs:sequence>
1842 415 <xs:element name="Precondition" type="tCondition" minOccurs="0"/>
1843 416 <xs:choice>
1844 417 <xs:element name="PlanModel">
1845 418 <xs:complexType>
1846 419 <xs:sequence>
1847 420 <xs:any namespace="##other" processContents="lax"/>
1848 421 </xs:sequence>
1849 422 </xs:complexType>
1850 423 </xs:element>
1851 424 <xs:element name="PlanModelReference">
1852 425 <xs:complexType>
1853 426 <xs:attribute name="reference" type="xs:anyURI"
1854 427 use="required"/>
1855 428 </xs:complexType>
1856 429 </xs:element>
1857 430 </xs:choice>
1858 431 </xs:sequence>
1859 432 <xs:attribute name="id" type="xs:ID" use="required"/>
1860 433 <xs:attribute name="name" type="xs:string" use="optional"/>
1861 434 <xs:attribute name="planType" type="xs:anyURI" use="required"/>
1862 435 <xs:attribute name="languageUsed" type="xs:anyURI" use="required"/>
1863 436 </xs:extension>
1864 437 </xs:complexContent>
1865 438 </xs:complexType>
1866 439
1867 440 <xs:complexType name="tPolicy">
1868 441 <xs:complexContent>
1869 442 <xs:extension base="tExtensibleElements">
1870 443 <xs:attribute name="name" type="xs:string" use="required"/>
1871 444 <xs:attribute name="type" type="xs:anyURI" use="required"/>
1872 445 </xs:extension>
1873 446 </xs:complexContent>
1874 447 </xs:complexType>
1875 448
1876 449 <xs:complexType name="tEnvironmentConstraint">
1877 450 <xs:sequence>
1878 451 <xs:any namespace="##other" processContents="lax"/>
1879 452 </xs:sequence>
1880 453 <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
1881 454 </xs:complexType>
1882 455
1883 456 <xs:complexType name="tExtensions">
1884 457 <xs:complexContent>
1885 458 <xs:extension base="tExtensibleElements">
1886 459 <xs:sequence>

```

```

1887 460     <xs:element name="Extension" type="tExtension"
1888 461         maxOccurs="unbounded"/>
1889 462     </xs:sequence>
1890 463 </xs:extension>
1891 464 </xs:complexContent>
1892 465 </xs:complexType>
1893 466
1894 467 <xs:complexType name="tExtension">
1895 468     <xs:complexContent>
1896 469         <xs:extension base="tExtensibleElements">
1897 470             <xs:attribute name="namespace" type="xs:anyURI" use="required"/>
1898 471             <xs:attribute name="mustUnderstand" type="tBoolean" use="optional"
1899 472                 default="yes"/>
1900 473         </xs:extension>
1901 474     </xs:complexContent>
1902 475 </xs:complexType>
1903 476
1904 477 <xs:complexType name="tParameter">
1905 478     <xs:attribute name="name" type="xs:string" use="required"/>
1906 479     <xs:attribute name="type" type="xs:string" use="required"/>
1907 480     <xs:attribute name="required" type="tBoolean" use="optional"
1908 481         default="yes"/>
1909 482 </xs:complexType>
1910 483
1911 484 <xs:complexType name="tWSDL">
1912 485     <xs:attribute name="portType" type="xs:QName" use="required"/>
1913 486     <xs:attribute name="operation" type="xs:NCName" use="optional"/>
1914 487 </xs:complexType>
1915 488
1916 489 <xs:complexType name="tOperation">
1917 490     <xs:complexContent>
1918 491         <xs:extension base="tExtensibleElements">
1919 492             <xs:sequence>
1920 493                 <xs:element name="Implementations">
1921 494                     <xs:complexType>
1922 495                         <xs:sequence>
1923 496                             <xs:element name="Implementation" maxOccurs="unbounded">
1924 497                                 <xs:complexType>
1925 498                                     <xs:choice>
1926 499                                         <xs:element name="ImplementationProper" type="xs:anyType"
1927 500                                             minOccurs="0"/>
1928 501                                         <xs:element name="ImplementationReference" minOccurs="0">
1929 502                                             <xs:complexType>
1930 503                                                 <xs:attribute name="ref" type="xs:anyURI"/>
1931 504                                             </xs:complexType>
1932 505                                         </xs:element>
1933 506                                     </xs:choice>
1934 507                                         <xs:attribute name="implementationID" type="xs:anyURI"/>
1935 508                                         <xs:attribute name="language" type="xs:anyURI"/>
1936 509                                     </xs:complexType>
1937 510                                         </xs:element>
1938 511                                     </xs:sequence>
1939 512                                 </xs:complexType>
1940 513                             </xs:element>
1941 514                             <xs:element name="InputParameters" minOccurs="0">
1942 515                                 <xs:complexType>
1943 516                                     <xs:sequence>
1944 517                                         <xs:element name="InputParameter" type="tParameter"

```

```

1945 518         maxOccurs="unbounded"/>
1946 519         </xs:sequence>
1947 520     </xs:complexType>
1948 521 </xs:element>
1949 522 <xs:element name="OutputParameters" minOccurs="0">
1950 523     <xs:complexType>
1951 524         <xs:sequence>
1952 525             <xs:element name="OutputParameter" type="tParameter"
1953 526                 maxOccurs="unbounded"/>
1954 527         </xs:sequence>
1955 528     </xs:complexType>
1956 529 </xs:element>
1957 530 </xs:sequence>
1958 531     <xs:attribute name="name" type="xs:NCName" use="required"/>
1959 532 </xs:extension>
1960 533 </xs:complexContent>
1961 534 </xs:complexType>
1962 535
1963 536 <xs:complexType name="tREST">
1964 537     <xs:attribute name="method" default="GET">
1965 538         <xs:simpleType>
1966 539             <xs:restriction base="xs:string">
1967 540                 <xs:enumeration value="GET"/>
1968 541                 <xs:enumeration value="PUT"/>
1969 542                 <xs:enumeration value="POST"/>
1970 543                 <xs:enumeration value="DELETE"/>
1971 544             </xs:restriction>
1972 545         </xs:simpleType>
1973 546     </xs:attribute>
1974 547     <xs:attribute name="requestURI" type="xs:anyURI" use="required"/>
1975 548     <xs:attribute name="requestPayload" type="xs:QName"/>
1976 549     <xs:attribute name="responsePayload" type="xs:QName"/>
1977 550 </xs:complexType>
1978 551
1979 552 <xs:complexType name="tCondition">
1980 553     <xs:sequence>
1981 554         <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1982 555     </xs:sequence>
1983 556     <xs:attribute name="expressionLanguage" type="xs:anyURI"
1984 557         use="required"/>
1985 558 </xs:complexType>
1986 559
1987 560 <xs:complexType name="tTopologyElementInstanceStates">
1988 561     <xs:sequence>
1989 562         <xs:element name="InstanceState" maxOccurs="unbounded">
1990 563             <xs:complexType>
1991 564                 <xs:attribute name="state" type="xs:anyURI" use="required"/>
1992 565             </xs:complexType>
1993 566         </xs:element>
1994 567     </xs:sequence>
1995 568 </xs:complexType>
1996 569
1997 570 <xs:simpleType name="tBoolean">
1998 571     <xs:restriction base="xs:string">
1999 572         <xs:enumeration value="yes"/>
2000 573         <xs:enumeration value="no"/>
2001 574     </xs:restriction>
2002 575 </xs:simpleType>

```

```
2003 576
2004 577 <xs:simpleType name="importedURI">
2005 578   <xs:restriction base="xs:anyURI"/>
2006 579 </xs:simpleType>
2007 580
2008 581 </xs:schema>
```

2009

Appendix D. Sample

2010

This appendix contains the full sample used in this specification.

2011

D.1 Sample Service Topology Definition

2012

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

2013

2014

2015

```
<Types>
```

2016

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

2017

```
  elementFormDefault="qualified"
```

2018

```
  attributeFormDefault="unqualified">
```

2019

```
<xs:element name="ApplicationProperties">
```

2020

```
<xs:complexType>
```

2021

```
<xs:sequence>
```

2022

```
<xs:element name="Owner" type="xs:string"/>
```

2023

```
<xs:element name="InstanceName" type="xs:string"/>
```

2024

```
<xs:element name="AccountID" type="xs:string"/>
```

2025

```
</xs:sequence>
```

2026

```
</xs:complexType>
```

2027

```
</xs:element>
```

2028

```
<xs:element name="AppServerProperties">
```

2029

```
<xs:complexType>
```

2030

```
<xs:sequence>
```

2031

```
<element name="HostName" type="string"/>
```

2032

```
<element name="IPAddress" type="string"/>
```

2033

```
<element name="HeapSize" type="positiveInteger"/>
```

2034

```
<element name="SoapPort" type="positiveInteger"/>
```

2035

```
</xs:sequence>
```

2036

```
</xs:complexType>
```

2037

```
</xs:element>
```

2038

```
</xs:schema>
```

2039

```
</Types>
```

2040

```
<TopologyTemplate id="SampleApplication">
```

2041

```
<NodeTemplate id="MyApplication"
```

2042

```
  name="My Application"
```

2043

```
  nodeType="abc:Application">
```

2044

```
<PropertyDefaults>
```

2045

```
<ApplicationProperties>
```

2046

```
<Owner>Frank</Owner>
```

2047

```
<InstanceName>Thomas' favorite application</InstanceName>
```

2048

```
</ApplicationProperties>
```

2049

```
</PropertyDefaults>
```

2050

```
<NodeTemplate/>
```

2051

```
<NodeTemplate id="MyAppServer"
```

2052

```
  name="My Application Server"
```

2053

```
  nodeType="abc:ApplicationServer"
```

2054

```
  minInstances="0"
```

2055

```
  maxInstances="unbounded"/>
```

2056

```
<RelationshipTemplate id="MyDeploymentRelationship"
```

2060


```

2061         relationshipType="deployedOn">
2062     <SourceElement id="MyApplication"/>
2063     <TargetElement id="MyAppServer"/>
2064 </RelationshipTemplate>
2065
2066 </TopologyTemplate>
2067
2068 <NodeTypes>
2069     <NodeType name="Application">
2070         <documentation xml:lang="EN">
2071             A reusable definition of a node type representing an
2072             application that can be deployed on application servers.
2073         </documentation>
2074         <NodeTypeProperties element="ApplicationProperties"/>
2075         <InstanceStates>
2076             <InstanceState state="http://www.my.com/started"/>
2077             <InstanceState state="http://www.my.com/stopped"/>
2078         </InstanceStates>
2079         <Interfaces>
2080             <Interface>
2081                 <Operation name="DeployApplication">
2082                     <InputParameters>
2083                         <InputParamter name="InstanceName"
2084                             type="string"/>
2085                         <InputParamter name="AppServerHostname"
2086                             type="string"/>
2087                         <InputParamter name="ContextRoot"
2088                             type="string"/>
2089                     </InputParameters>
2090                     <Implementations>
2091                         <Implementation>
2092                             ...
2093                         </Implementation>
2094                     </Implementations>
2095                 </Operation>
2096             </Interface>
2097         </Interfaces>
2098     </NodeType>
2099     <NodeType name="ApplicationServer"
2100         targetNamespace="http://www.ibm.com/sample">
2101         <NodeTypeProperties element="AppServerProperties"/>
2102         <Interfaces>
2103             <Interface>
2104                 <Operation name="AcquireNetworkAddress">
2105                     <OutputParameters>
2106                         <OutputParamter name="Hostname"
2107                             type="string"/>
2108                         <OutputParamter name="IPAddress"
2109                             type="string"/>
2110                     </OutputParameters>
2111                     <Implementations>
2112                         <Implementation>
2113                             ...
2114                         </Implementation>
2115                     </Implementations>
2116                 </Operation>
2117                 <Operation name="DeployApplicationServer">
2118                     <InputParameters>

```

```

2119         <InputParamter name="Hostname"
2120             type="string"/>
2121         <InputParamter name="IPAddress"
2122             type="string"/>
2123         <InputParamter name="HeapSize"
2124             type="int"/>
2125         <InputParamter name="SoapPort"
2126             type="int"/>
2127     </InputParameters>
2128     <OutputParameters>
2129         <OutputParamter name="ServerID"
2130             type="string"/>
2131     </OutputParameters>
2132     <Implementations>
2133         <Implementation>
2134             ...
2135         </Implementation>
2136     </Implementations>
2137 </Operation>
2138 </Interface>
2139 </Interfaces>
2140 </NodeType>
2141 </NodeTypes>
2142
2143 <RelationshipTypes>
2144     <documentation xml:lang="EN">
2145         A reusable definition of relation that expresses deployment of
2146         an artifact on a hosting environment.
2147     </documentation>
2148     <RelationshipType name="deployedOn"
2149         semantics="www.my.com/RelSemantics/deployedOn">
2150     </RelationshipType>
2151 </RelationshipTypes>
2152
2153 <Plans>
2154     <Plan id="DeployApplication"
2155         name="Sample Application Build Plan"
2156         planType="http://docs.oasis-
2157             open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
2158         languageUsed="http://www.omg.org/spec/BPMN/2.0/">
2159
2160         <PreCondition expressionLanguage="www.my.com/text">?
2161             Run only if funding is available
2162         </PreCondition>
2163
2164     <PlanModel>
2165         <process name="DeployNewApplication" id="p1">
2166             <documentation>This process deploys a new instance of the
2167             sample application.
2168             </documentation>
2169
2170             <task id="t1" name="CreateAccount"/>
2171
2172             <task id="t2" name="AcquireNetworkAddresses"
2173                 isSequential="false"
2174                 loopDataInput="t2Input.LoopCounter"/>
2175             <documentation>Assumption: t2 gets data of type "input"
2176                 as input and this data has a field names "LoopCounter"

```

```
2177         that contains the actual multiplicity of the task.
2178     </documentation>
2179
2180     <task id="t3" name="DeployApplicationServer"
2181         isSequential="false"
2182         loopDataInput="t3Input.LoopCounter"/>
2183
2184     <task id="t4" name="DeployApplication"
2185         isSequential="false"
2186         loopDataInput="t4Input.LoopCounter"/>
2187
2188     <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
2189     <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
2190     <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
2191 </process>
2192 </PlanModel>
2193 </Plan>
2194
2195 <Plan id="RemoveApplication"
2196     planType="http://docs.oasis-
2197     open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
2198     languageUsed="http://docs.oasis-
2199     open.org/wsbpel/2.0/process/executable">
2200     <PlanModelReference reference="prj:RemoveApp"/>
2201 </Plan>
2202 </Plans>
2203
2204 </ServiceTemplate>
```

2205

Appendix E. Revision History

2206

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.

2207

2208