Topology and Orchestration Specification for Cloud Applications Version 1.0

Committee Specification Draft 08

09 May 2013

Specification URIs

This version:

http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd08/TOSCA-v1.0-csd08.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd08/TOSCA-v1.0-csd08.html
http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd08/TOSCA-v1.0-csd08.doc

Previous version:

http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html
http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.doc

Latest version:

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

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:

Derek Palma (dpalma@vnomic.com), Vnomic
Thomas Spatzier (thomas.spatzier@de.ibm.com), IBM

Additional artifacts:

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

- XML schemas: http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd08/schemas/

Declared XML namespace:

- 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. 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.

 Copyright © OASIS Open 2013. All Rights Reserved.

Status:

This document was last revised or approved by the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/tosca/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/tosca/ipr.php).

Citation format:

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

[TOSCA-v1.0]

Topology and Orchestration Specification for Cloud Applications Version 1.0. 09 May 2013. OASIS Committee Specification Draft 08. http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd08/TOSCA-v1.0-csd08.html.

Copyright © OASIS Open 2013. All Rights Reserved.

Notices

Copyright © OASIS Open 2013. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1 Introduction 7
2 Language Design 8
2.1 Dependencies on Other Specifications 8
2.2 Notational Conventions 8
2.3 Normative References 8
2.4 Non-Normative References 8
2.5 Typographical Conventions 9
2.6 Namespaces 9
2.7 Language Extensibility 10
3 Core Concepts and Usage Pattern 11
3.1 Core Concepts 11
3.2 Use Cases 12
3.2.1 Services as Marketable Entities 12
3.2.2 Portability of Service Templates 13
3.2.3 Service Composition 13
3.2.4 Relation to Virtual Images 13
3.3 Service Templates and Artifacts 13
3.4 Requirements and Capabilities 14
3.5 Composition of Service Templates 15
3.6 Policies in TOSCA 15
3.7 Archive Format for Cloud Applications 16
4 The TOSCA Definitions Document 18
4.1 XML Syntax 18
4.2 Properties 19
4.3 Example 22
5 Service Templates 23
5.1 XML Syntax 23
5.2 Properties 26
5.3 Example 37
6 Node Types 39
6.1 XML Syntax 39
6.2 Properties 40
6.3 Derivation Rules 43
6.4 Example 43
7 Node Type Implementations 45
7.1 XML Syntax 45
7.2 Properties 46
7.3 Derivation Rules 48
7.4 Example 49
8 Relationship Types 50
8.1 XML Syntax 50
8.2 Properties 51
8.3 Derivation Rules 52

Copyright © OASIS Open 2013. All Rights Reserved.

8.4 Example 53
9 Relationship Type Implementations 54
9.1 XML Syntax 54
9.2 Properties 54
9.3 Derivation Rules 56
9.4 Example 57
10 Requirement Types 58
10.1 XML Syntax 58
10.2 Properties 58
10.3 Derivation Rules 59
10.4 Example 60
11 Capability Types 61
11.1 XML Syntax 61
11.2 Properties 61
11.3 Derivation Rules 62
11.4 Example 62
12 Artifact Types 64
12.1 XML Syntax 64
12.2 Properties 64
12.3 Derivation Rules 65
12.4 Example 65
13 Artifact Templates 67
13.1 XML Syntax 67
13.2 Properties 67
13.3 Example 69
14 Policy Types 70
14.1 XML Syntax 70
14.2 Properties 70
14.3 Derivation Rules 71
14.4 Example 72
15 Policy Templates 73
15.1 XML Syntax 73
15.2 Properties 73
15.3 Example 74
16 Cloud Service Archive (CSAR) 75
16.1 Overall Structure of a CSAR 75
16.2 TOSCA Meta File 75
16.3 Example 76
17 Security Considerations 80
18 Conformance 81
Appendix A. Portability and Interoperability Considerations 82
Appendix B. Acknowledgements 83
Appendix C. Complete TOSCA Grammar 85
Appendix D. TOSCA Schema 93
Appendix E. Sample 109

 Copyright © OASIS Open 2013. All Rights Reserved.

E.1 Sample Service Topology Definition 109
Appendix F. Revision History 112

 Copyright © OASIS Open 2013. All Rights Reserved.

1 Introduction

Cloud computing can become more valuable if the semi-automatic creation and management of application layer services can be ported across alternative cloud implementation environments so that the services remain interoperable. This core TOSCA specification provides a language to describe service components and their relationships using a *service topology*, and it provides for describing the management procedures that create or modify services using *orchestration processes*. The combination of topology and orchestration in a *Service Template* describes what is needed to be preserved across deployments in different environments to enable interoperable deployment of cloud services and their management throughout the complete lifecycle (e.g. scaling, patching, monitoring, etc.) when the applications are ported over alternative cloud environments.

 Copyright © OASIS Open 2013. All Rights Reserved.

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:

- XML Schema 1.0

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].

This specification follows XML naming and design rules as described in **[UNCEFACT XMLNDR]**, i.e. uses upper camel-case notation for XML element names and lower camel-case notation for XML attribute names.

2.3 Normative References

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

[RFC 2396] Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, available via http://www.faqs.org/rfcs/rfc2396.html

[XML Base] XML Base (Second Edition), W3C Recommendation, http://www.w3.org/TR/xmlbase/

[XML Infoset] XML Information Set, W3C Recommendation, http://www.w3.org/TR/2001/REC-xml-infoset-20011024/

[XML Namespaces] Namespaces in XML 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/REC-xml-names/

[XML Schema Part 1] XML Schema Part 1: Structures, W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-1/

[XML Schema Part 2] XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-2/

[XMLSpec] XML Specification, W3C Recommendation, February 1998, http://www.w3.org/TR/1998/REC-xml-19980210

2.4 Non-Normative References

[BPEL 2.0] *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. 11 April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[BPMN 2.0] OMG Business Process Model and Notation (BPMN) Version 2.0, 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

 Copyright © OASIS Open 2013. All Rights Reserved.

53 **[XPATH 1.0]** XML Path Language (XPath) Version 1.0, W3C Recommendation, November
54 1999, http://www.w3.org/TR/1999/REC-xpath-19991116
55 **[UNCEFACT XMLNDR]** UN/CEFACT XML Naming and Design Rules Technical Specification,
56 Version 3.0,
57 http://www.unece.org/fileadmin/DAM/cefact/xml/UNCEFACT+XML+NDR+V3p0.p
58 df
59

60 2.5 Typographical Conventions

61 This specification uses the following conventions inside tables describing the resource data model:

- 62 Resource names, and any other name that is usable as a type (i.e., names of embedded
63 structures as well as atomic types such as "integer", "string"), are in italic.
- 64 Attribute names are in regular font.

65 In addition, this specification uses the following syntax to define the serialization of resources:

- 66 Values in italics indicate data types instead of literal values.
- 67 Characters are appended to items to indicate cardinality:
  - 68 "?" (0 or 1)
  - 69 "*" (0 or more)
  - 70 "+" (1 or more)
- 71 Vertical bars, "|", denote choice. For example, "a|b" means a choice between "a" and "b".
- 72 Parentheses, "(" and ")", are used to indicate the scope of the operators "?", "*", "+" and "|".
- 73 Ellipses (i.e., "...") indicate points of extensibility. Note that the lack of an ellipses does not mean
74 no extensibility point exists, rather it is just not explicitly called out - usually for the sake of brevity.

75 2.6 Namespaces

76 This specification uses a number of namespace prefixes throughout; they are listed in Table 1**.** Note that
77 the choice of any namespace prefix is arbitrary and not semantically significant (see [XML Namespaces]).
78 Furthermore, the namespace http://docs.oasis-open.org/tosca/ns/2011/12 is assumed to be the default
79 namespace, i.e. the corresponding namespace name tosca is omitted in this specification to improve
80 readability.
81

Prefix	**Namespace**
tosca	http://docs.oasis-open.org/tosca/ns/2011/12
xs	http://www.w3.org/2001/XMLSchema

82 Table 1: Prefixes and namespaces used in this specification
83

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

 Copyright © OASIS Open 2013. All Rights Reserved.

87 ## 2.7 Language Extensibility

88 The TOSCA extensibility mechanism allows:

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

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

 Copyright © OASIS Open 2013. All Rights Reserved.

3 Core Concepts and Usage Pattern

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

3.1 Core Concepts

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

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

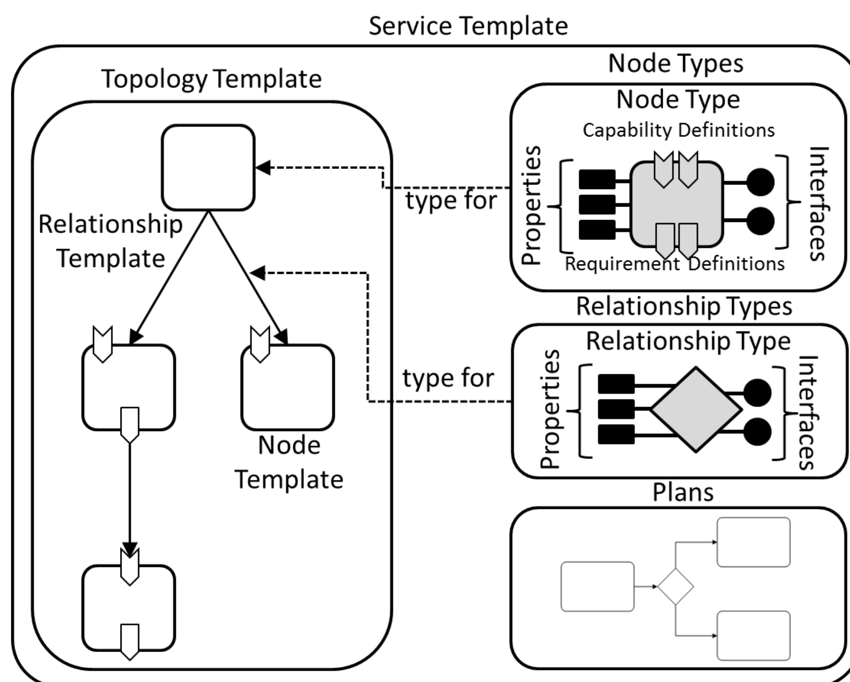


Figure 1: Structural Elements of a Service Template and their Relations

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

 Copyright © OASIS Open 2013. All Rights Reserved.

125 A *Relationship Template* specifies the occurrence of a relationship between nodes in a Topology
126 Template. Each Relationship Template refers to a Relationship Type that defines the semantics and any
127 properties of the relationship. Relationship Types are defined separately for reuse purposes. The
128 Relationship Template indicates the elements it connects and the direction of the relationship by defining
129 one source and one target element (in nested `SourceElement` and `TargetElement` elements). The
130 Relationship Template also defines any constraints with the OPTIONAL
131 `RelationshipConstraints` element.

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

135 A deployed service is an instance of a Service Template. More precisely, the instance is derived by
136 instantiating the Topology Template of its Service Template, most often by running a special plan defined
137 for the Service Template, often referred to as build plan. The build plan will provide actual values for the
138 various properties of the various Node Templates and Relationship Templates of the Topology Template.
139 These values can come from input passed in by users as triggered by human interactions defined within
140 the build plan, by automated operations defined within the build plan (such as a directory lookup), or the
141 templates can specify default values for some properties. The build plan will typically make use of
142 operations of the Node Types of the Node Templates.

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

149 *Plans* defined in a Service Template describe the management aspects of service instances, especially
150 their creation and termination. These plans are defined as process models, i.e. a workflow of one or more
151 steps. Instead of providing another language for defining process models, the specification relies on
152 existing languages like BPMN or BPEL. Relying on existing standards in this space facilitates portability
153 and interoperability, but any language for defining process models can be used. The TOSCA metamodel
154 provides containers to either refer to a process model (via *Plan Model Reference*) or to include the actual
155 model in the plan (via *Plan Model*). A process model can contain tasks (using BPMN terminology) that
156 refer to operations of Interfaces of Node Templates (or operations defined by the Node Types specified in
157 the `type` attribute of the Node Templates, respectively), operations of Interfaces of Relationship
158 Templates (or operations defined by the Relationship Types specified in the `type` attribute of the
159 Relationship Templates, respectively), or any other interface (e.g. the invocation of an external service for
160 licensing); in doing so, a plan can directly manipulate nodes of the topology of a service or interact with
161 external systems.

162 3.2 Use Cases

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

164 3.2.1 Services as Marketable Entities

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

173 Making a concrete instance of a Topology Template can be done by running a corresponding Plan (so-
174 called instantiating management plan, a.k.a. build plan). This build plan could be provided by the service
175 developer who also creates the Service Template. The build plan can be adapted to the concrete

 Copyright © OASIS Open 2013. All Rights Reserved.

176 environment of a particular service provider. Other management plans useful in various states of the
177 whole lifecycle of a service could be specified as part of a Service Template. Similar to build plans such
178 management plans can be adapted to the concrete environment of a particular service provider.

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

186 3.2.2 Portability of Service Templates

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

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

195 3.2.3 Service Composition

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

202 3.2.4 Relation to Virtual Images

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

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

215 3.3 Service Templates and Artifacts

216 An artifact represents the content needed to realize a deployment such as an executable (e.g. a script, an
217 executable program, an image), a configuration file or data file, or something that might be needed so that
218 another executable can run (e.g. a library). Artifacts can be of different types, for example EJBs or python
219 scripts. The content of an artifact depends on its type. Typically, descriptive metadata will also be
220 provided along with the artifact. This metadata might be needed to properly process the artifact, for
221 example by describing the appropriate execution environment.

222 TOSCA distinguishes two kinds of artifacts: *implementation artifacts* and *deployment artifacts*. An
223 implementation artifact represents the executable of an operation of a node type, and a deployment

 Copyright © OASIS Open 2013. All Rights Reserved.

artifact represents the executable for materializing instances of a node. For example, a REST operation to store an image can have an implementation artifact that is a WAR file. The node type this REST operation is associated with can have the image itself as a deployment artifact.

The fundamental difference between implementation artifacts and deployment artifacts is twofold, namely

1. the point in time when the artifact is deployed, and
2. by what entity and to where the artifact is deployed.

The operations of a node type perform management actions on (instances of) the node type. The implementations of such operations can be provided as implementation artifacts. Thus, the implementation artifacts of the corresponding operations have to be deployed in the management environment before any management operation can be started. In other words, "a TOSCA supporting environment" (i.e. a so-called TOSCA container) MUST be able to process the set of implementation artifacts types needed to execute those management operations. One such management operation could be the instantiation of a node type.

The instantiation of a node type can require providing deployment artifacts in the target managed environment. For this purpose, a TOSCA container supports a set of types of deployment artifacts that it can process. A service template that contains (implementation or deployment) artifacts of non-supported types cannot be processed by the container (resulting in an error during import).

3.4 Requirements and Capabilities

TOSCA allows for expressing *requirements* and *capabilities* of components of a service. This can be done, for example, to express that one component depends on (requires) a feature provided by another component, or to express that a component has certain requirements against the hosting environment such as for the allocation of certain resources or the enablement of a specific mode of operation.

Requirements and capabilities are modeled by annotating Node Types with *Requirement Definitions* and *Capability Definitions* of certain types. *Requirement Types* and *Capability Types* are defined as reusable entities so that those definitions can be used in the context of several Node Types. For example, a Requirement Type "DatabaseConnectionRequirement" might be defined to describe the requirement of a client for a database connection. This Requirement Type can then be reused for all kinds of Node Types that represent, for example, application with the need for a database connection.

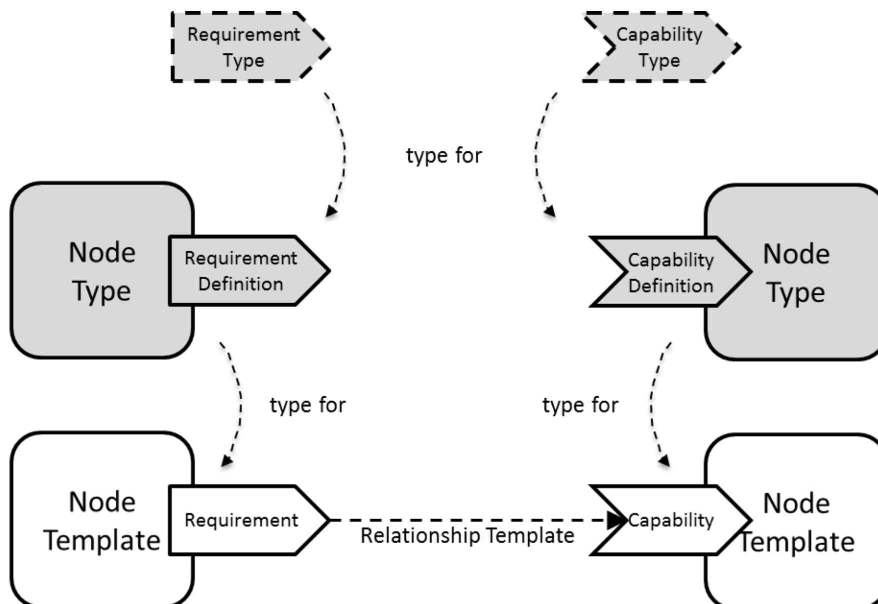


Figure 2: Requirements and Capabilities

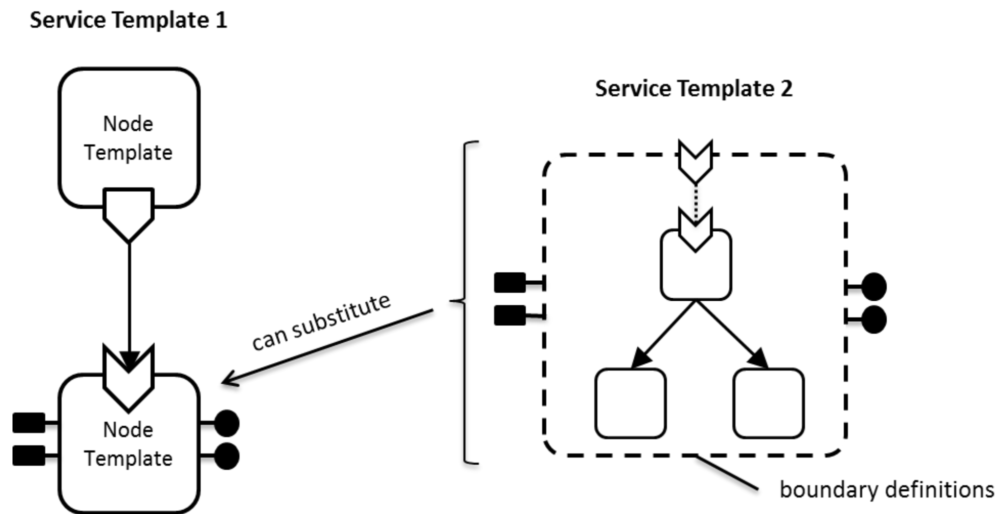
Copyright © OASIS Open 2013. All Rights Reserved.

256 Node Templates which have corresponding Node Types with Requirement Definitions or Capability
257 Definitions will include representations of the respective *Requirements* and *Capabilities* with content
258 specific to the respective Node Template. For example, while Requirement Types just represent
259 Requirement metadata, the Requirement represented in a Node Template can provide concrete values
260 for properties defined in the Requirement Type. In addition, Requirements and Capabilities of Node
261 Templates in a Topology Template can optionally be connected via Relationship Templates to indicate
262 that a specific requirement of one node is fulfilled by a specific capability provided by another node.

263 Requirements can be matched in two ways as briefly indicated above: (1) requirements of a Node
264 Template can be matched by capabilities of another Node Template in the same Service Template by
265 connecting the respective requirement-capability-pairs via Relationship Templates; (2) requirements of a
266 Node Template can be matched by the general hosting environment (or the TOSCA container), for
267 example by allocating needed resources for a Node Template during instantiation.

268 3.5 Composition of Service Templates

269 Service Templates can be based on and built on-top of other Service Templates based on the concept of
270 Requirements and Capabilities introduced in the previous section. For example, a Service Template for a
271 business application that is hosted on an application server tier might focus on defining the structure and
272 manageability behavior of the application itself. The structure of the application server tier hosting the
273 application can be provided in a separate Service Template built by another vendor specialized in
274 deploying and managing application servers. This approach enables separation of concerns and re-use of
275 common infrastructure templates.



276

277 Figure 3: Service Template Composition

278 From the point of view of a Service Template (e.g. the business application Service Template from the
279 example above) that uses another Service Template, the other Service Template (e.g. the application
280 server tier) “looks” like just a Node Template. During deployment, however, this Node Template can be
281 substituted by the second Service Template if it exposes the same boundaries (i.e. properties,
282 capabilities, etc.) as the Node Template. Thus, a substitution with any Service Template that has the
283 same *boundary definitions* as a certain Node Template in one Service Template becomes possible,
284 allowing for a flexible composition of different Service Templates. This concept also allows for providing
285 substitutable alternatives in the form of Service Templates. For example, a Service Template for a single
286 node application server tier and a Service Template for a clustered application server tier might exist, and
287 the appropriate option can be selected per deployment.

288 3.6 Policies in TOSCA

289 Non-functional behavior or quality-of-services are defined in TOSCA by means of policies. A Policy can
290 express such diverse things like monitoring behavior, payment conditions, scalability, or continuous
291 availability, for example.

 Copyright © OASIS Open 2013. All Rights Reserved.

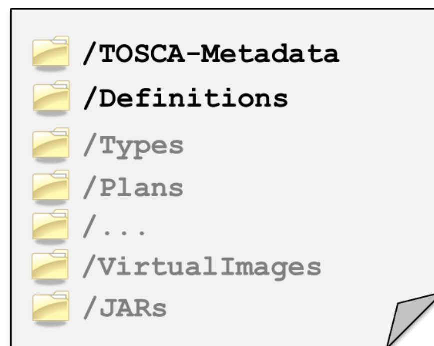
292 A Node Template can be associated with a set of Policies collectively expressing the non-functional
293 behavior or quality-of-services that each instance of the Node Template will expose. Each Policy specifies
294 the actual properties of the non-functional behavior, like the concrete payment information (payment
295 period, currency, amount etc) about the individual instances of the Node Template.

296 These properties are defined by a Policy Type. Policy Types might be defined in hierarchies to properly
297 reflect the structure of non-functional behavior or quality-of-services in particular domains. Furthermore, a
298 Policy Type might be associated with a set of Node Types the non-functional behavior or quality-of-
299 service it describes.

300 Policy Templates provide actual values of properties of the types defined by Policy Types. For example, a
301 Policy Template for monthly payments for US customers will set the "payment period" property to
302 "monthly" and the "currency" property to "US$", leaving the "amount" property open. The "amount"
303 property will be set when the corresponding Policy Template is used for a Policy within a Node Template.
304 Thus, a Policy Template defines the invariant properties of a Policy, while the Policy sets the variant
305 properties resulting from the actual usage of a Policy Template in a Node Template.

306 3.7 Archive Format for Cloud Applications

307 In order to support in a certain environment the execution and management of the lifecycle of a cloud
308 application, all corresponding artifacts have to be available in that environment. This means that beside
309 the service template of the cloud application, the deployment artifacts and implementation artifacts have
310 to be available in that environment. To ease the task of ensuring the availability of all of these, this
311 specification defines a corresponding archive format called CSAR (Cloud Service ARchive).



312

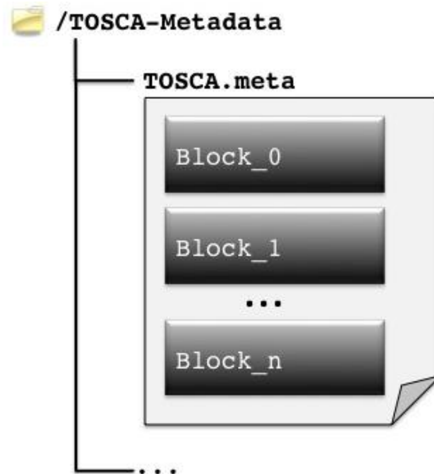
313 Figure 4: Structure of the CSAR

314 A CSAR is a container file, i.e. it contains multiple files of possibly different file types. These files are
315 typically organized in several subdirectories, each of which contains related files (and possibly other
316 subdirectories etc). The organization into subdirectories and their content is specific for a particular cloud
317 application. CSARs are zip files, typically compressed.

318 Each CSAR MUST contain a subdirectory called *TOSCA-Metadata*. This subdirectory MUST contain a
319 so-called *TOSCA meta file*. This file is named `TOSCA` and has the file extension `.meta`. It represents
320 metadata of the other files in the CSAR. This metadata is given in the format of name/value pairs. These
321 name/value pairs are organized in blocks. Each block provides metadata of a certain artifact of the CSAR.
322 An empty line separates the blocks in the TOSCA meta file.

 Copyright © OASIS Open 2013. All Rights Reserved.

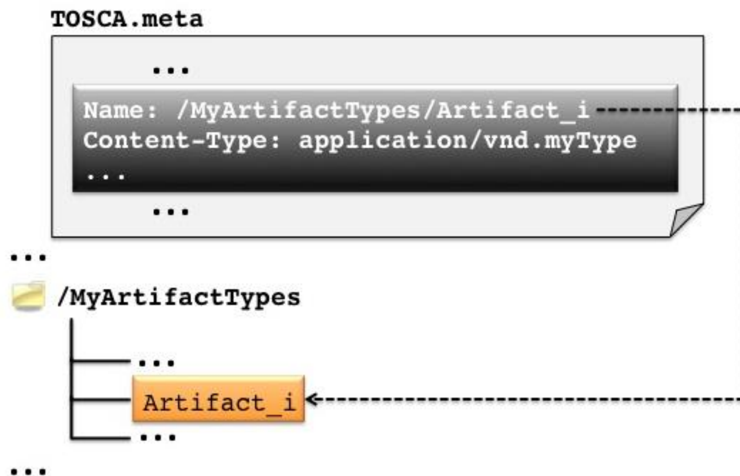
323



324 Figure 5: Structure of the TOSCA Meta File

325 The first block of the TOSCA meta file (Block_0 in Figure 5) provides metadata of the CSAR itself (e.g. its
326 version, creator etc). Each other block begins with a name/value pair that points to an artifact within the
327 CSAR by means of a pathname. The remaining name/value pairs in a block are the proper metadata of
328 the pointed to artifact. For example, a corresponding name/value pair specifies the MIME-type of the
329 artifact.

330



331 Figure 6: Providing Metadata for Artifacts

332

 Copyright © OASIS Open 2013. All Rights Reserved.

4 The TOSCA Definitions Document

All elements needed to define a TOSCA Service Template – such as Node Type definitions, Relationship Type definitions, etc. – as well as Service Templates themselves are provided in TOSCA *Definitions* documents. This section explains the overall structure of a TOSCA Definitions document, the extension mechanism, and import features. Later sections describe in detail Service Templates, Node Types, Node Type Implementations, Relationship Types, Relationship Type Implementations, Requirement Types, Capability Types, Artifact Types, Artifact Templates, Policy Types and Policy Templates.

4.1 XML Syntax

The following pseudo schema defines the XML syntax of a Definitions document:

```
01 <Definitions id="xs:ID"
02              name="xs:string"?
03              targetNamespace="xs:anyURI">
04
05   <Extensions>
06     <Extension namespace="xs:anyURI"
07                mustUnderstand="yes|no"?/> +
08   </Extensions> ?
09
10   <Import namespace="xs:anyURI"?
11           location="xs:anyURI"?
12           importType="xs:anyURI"/> *
13
14   <Types>
15     <xs:schema .../> *
16   </Types> ?
17
18   (
19     <ServiceTemplate> ... </ServiceTemplate>
20   |
21     <NodeType> ... </NodeType>
22   |
23     <NodeTypeImplementation> ... </NodeTypeImplementation>
24   |
25     <RelationshipType> ... </RelationshipType>
26   |
27     <RelationshipTypeImplementation> ... </RelationshipTypeImplementation>
28   |
29     <RequirementType> ... </RequirementType>
30   |
31     <CapabilityType> ... </CapabilityType>
32   |
33     <ArtifactType> ... </ArtifactType>
34   |
35     <ArtifactTemplate> ... </ArtifactTemplate>
36   |
37     <PolicyType> ... </PolicyType>
38   |
39     <PolicyTemplate> ... </PolicyTemplate>
40   ) +
41
42 </Definitions>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

4.2 Properties

The `Definitions` element has the following properties:

- `id`: This attribute specifies the identifier of the Definitions document which MUST be unique within the target namespace.
- `name`: This OPTIONAL attribute specifies a descriptive name of the Definitions document.
- `targetNamespace`: The value of this attribute specifies the target namespace for the Definitions document. All elements defined within the Definitions document will be added to this namespace unless they override this attribute by means of their own `targetNamespace` attributes.
- `Extensions`: This OPTIONAL element specifies namespaces of TOSCA extension attributes and extension elements. If present, the `Extensions` element MUST include at least one `Extension` element.
  The Extension element has the following properties:
  - `namespace`: This attribute specifies the namespace of TOSCA extension attributes and extension elements.
  - `mustUnderstand`: This OPTIONAL attribute specifies whether the extension MUST be understood by a compliant implementation. If the `mustUnderstand` attribute has value “yes” (which is the default value for this attribute) the extension is mandatory. Otherwise, the extension is optional.
    If a TOSCA implementation does not support one or more of the mandatory extensions, then the Definitions document MUST be rejected. Optional extensions MAY be ignored. It is not necessary to declare optional extensions.
    The same extension URI MAY be declared multiple times in the `Extensions` element. If an extension URI is identified as mandatory in one `Extension` element and optional in another, then the mandatory semantics have precedence and MUST be enforced. The extension declarations in an `Extensions` element MUST be treated as an unordered set.
- `Import`: This element declares a dependency on external TOSCA Definitions, XML Schema definitions, or WSDL definitions. Any number of `Import` elements MAY appear as children of the `Definitions` element.
  The Import element has the following properties:
  - `namespace`: This OPTIONAL attribute specifies an absolute URI that identifies the imported definitions. An `Import` element without a `namespace` attribute indicates that external definitions are in use, which are not namespace-qualified. If a `namespace` attribute is specified then the imported definitions MUST be in that namespace. If no namespace is specified then the imported definitions MUST NOT contain a `targetNamespace` specification. The namespace http://www.w3.org/2001/XMLSchema is imported implicitly. Note, however, that there is no implicit XML Namespace prefix defined for http://www.w3.org/2001/XMLSchema.
  - `location`: This OPTIONAL attribute contains a URI indicating the location of a document that contains relevant definitions. The location URI MAY be a relative URI, following the usual rules for resolution of the URI base [XML Base, RFC 2396]. An `Import` element without a `location` attribute indicates that external definitions are used but makes no statement about where those definitions might be found. The `location` attribute is a hint and a TOSCA compliant implementation is not obliged to retrieve the document being imported from the specified location.

 Copyright © OASIS Open 2013. All Rights Reserved.

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

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

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

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

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

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

- `ServiceTemplate`: This element specifies a complete Service Template for a cloud application. A Service Template contains a definition of the Topology Template of the cloud application, as well as any number of Plans. Within the Service Template, any type definitions (e.g. Node Types, Relationship Types, etc.) defined in the same Definitions document or in imported Definitions document can be used.

- `NodeType`: This element specifies a type of Node that can be referenced as a type for Node Templates of a Service Template.

- `NodeTypeImplementation`: This element specifies the implementation of the manageability behavior of a type of Node that can be referenced as a type for Node Templates of a Service Template.

- `RelationshipType`: This element specifies a type of Relationship that can be referenced as a type for Relationship Templates of a Service Template.

 Copyright © OASIS Open 2013. All Rights Reserved.

- `RelationshipTypeImplementation`: This element specifies the implementation of the manageability behavior of a type of Relationship that can be referenced as a type for Relationship Templates of a Service Template.
- `RequirementType`: This element specifies a type of Requirement that can be exposed by Node Types used in a Service Template.
- `CapabilityType`: This element specifies a type of Capability that can be exposed by Node Types used in a Service Template.
- `ArtifactType`: This element specifies a type of artifact used within a Service Template. Artifact Types might be, for example, application modules such as .war files or .ear files, operating system packages like RPMs, or virtual machine images like .ova files.
- `ArtifactTemplate`: This element specifies a template describing an artifact referenced by parts of a Service Template. For example, the installable artifact for an application server node might be defined as an artifact template.
- `PolicyType`: This element specifies a type of Policy that can be associated to Node Templates defined within a Service Template. For example, a scaling policy for nodes in a web server tier might be defined as a Policy Type, which specifies the attributes the scaling policy can have.
- `PolicyTemplate`: This element specifies a template of a Policy that can be associated to Node Templates defined within a Service Template. Other than a Policy Type, a Policy Template can define concrete values for a policy according to the set of attributes specified by the Policy Type the Policy Template refers to.

A TOSCA Definitions document MUST define at least one of the elements `ServiceTemplate`, `NodeType`, `NodeTypeImplementation`, `RelationshipType`, `RelationshipTypeImplementation`, `RequirementType`, `CapabilityType`, `ArtifactType`, `ArtifactTemplate`, `PolicyType`, or `PolicyTemplate`, but it can define any number of those elements in an arbitrary order.

This technique supports a modular definition of Service Templates. For example, one Definitions document can contain only Node Type and Relationship Type definitions that can then be imported into another Definitions document that only defines a Service Template using those Node Types and Relationship Types. Similarly, Node Type Properties can be defined in separate XML Schema Definitions that are imported and referenced when defining a Node Type.

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

```
01 <documentation source="xs:anyURI"? xml:lang="xs:language"?>
02   ...
03 </documentation>
```

Example of use of a `documentation` element:

```
01 <Definitions id="MyDefinitions" name="My Definitions" ...>
02
03   <documentation xml:lang="EN">
04     This is a simple example of the usage of the documentation
05     element nested under a Definitions element. It could be used,
06     for example, to describe the purpose of the Definitions document
07     or to give an overview of elements contained within the Definitions
08     document.
09   </documentation>
10
11 </Definitions>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

527 ## 4.3 Example

528 The following Definitions document defines two Node Types, “Application” and “ApplicationServer”, as
529 well as one Relationship Type “ApplicationHostedOnApplicationServer”. The properties definitions for the
530 two Node Types are specified in a separate XML schema definition file which is imported into the
531 Definitions document by means of the `Import` element.

```
532 01 <Definitions id="MyDefinitions" name="My Definitions"
533 02   targetNamespace="http://www.example.com/MyDefinitions"
534 03   xmlns:my="http://www.example.com/MyDefinitions">
535 04
536 05   <Import importType="http://www.w3.org/2001/XMLSchema"
537 06     namespace="http://www.example.com/MyDefinitions">
538 07
539 08   <NodeType name="Application">
540 09     <PropertiesDefinition element="my:ApplicationProperties"/>
541 10   </NodeType>
542 11
543 12   <NodeType name="ApplicationServer">
544 13     <PropertiesDefinition element="my:ApplicationServerProperties"/>
545 14   </NodeType>
546 15
547 16   <RelationshipType name="ApplicationHostedOnApplicationServer">
548 17     <ValidSource typeRef="my:Application"/>
549 18     <ValidTarget typeRef="my:ApplicationServer"/>
550 19   </RelationshipTemplate>
551 20
552 21 </Definitions>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

5 Service Templates

This chapter specifies how *Service Templates* are defined. A Service Template describes the structure of a cloud application by means of a Topology Template, and it defines the manageability behavior of the cloud application in the form of Plans.

Elements within a Service Template, such as Node Templates defined in the Topology Template, refer to other TOSCA element, such as Node Types that can be defined in the same Definitions document containing the Service Template, or that can be defined in separate, imported Definitions documents.

Service Templates can be defined for being directly used for the deployment and management of a cloud application, or they can be used for composition into larger Service Template (see section 3.5 for details).

5.1 XML Syntax

The following pseudo schema defines the XML syntax of a Service Template:

```
01 <ServiceTemplate id="xs:ID"
02                  name="xs:string"?
03                  targetNamespace="xs:anyURI"
04                  substitutableNodeType="xs:QName"?>
05
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <BoundaryDefinitions>
11     <Properties>
12       XML fragment
13       <PropertyMappings>
14         <PropertyMapping serviceTemplatePropertyRef="xs:string"
15                          targetObjectRef="xs:IDREF"
16                          targetPropertyRef="xs:string"/> +
17         </PropertyMappings/> ?
18     </Properties> ?
19
20     <PropertyConstraints>
21       <PropertyConstraint property="xs:string"
22                           constraintType="xs:anyURI"> +
23         constraint ?
24       </PropertyConstraint>
25     </PropertyConstraints> ?
26
27     <Requirements>
28       <Requirement name="xs:string"? ref="xs:IDREF"/> +
29     </Requirements> ?
30
31     <Capabilities>
32       <Capability name="xs:string"? ref="xs:IDREF"/> +
33     </Capabilities> ?
34
35     <Policies>
36       <Policy name="xs:string"? policyType="xs:QName"
37               policyRef="xs:QName"?>
38         policy specific content ?
39       </Policy> +
40     </Policies> ?
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
604  41
605  42     <Interfaces>
606  43       <Interface name="xs:NCName">
607  44         <Operation name="xs:NCName">
608  45           (
609  46             <NodeOperation nodeRef="xs:IDREF"
610  47                            interfaceName="xs:anyURI"
611  48                            operationName="xs:NCName"/>
612  49           |
613  50             <RelationshipOperation relationshipRef="xs:IDREF"
614  51                                    interfaceName="xs:anyURI"
615  52                                    operationName="xs:NCName"/>
616  53           |
617  54             <Plan planRef="xs:IDREF"/>
618  55           )
619  56         </Operation> +
620  57       </Interface> +
621  58     </Interfaces> ?
622  59
623  60   </BoundaryDefinitions> ?
624  61
625  62   <TopologyTemplate>
626  63     (
627  64       <NodeTemplate id="xs:ID" name="xs:string"? type="xs:QName"
628  65                     minInstances="xs:integer"?
629  66                     maxInstances="xs:integer | xs:string"?>
630  67         <Properties>
631  68           XML fragment
632  69         </Properties> ?
633  70
634  71         <PropertyConstraints>
635  72           <PropertyConstraint property="xs:string"
636  73                               constraintType="xs:anyURI">
637  74             constraint ?
638  75           </PropertyConstraint> +
639  76         </PropertyConstraints> ?
640  77
641  78         <Requirements>
642  79           <Requirement id="xs:ID" name="xs:string" type="xs:QName"> +
643  80             <Properties>
644  81               XML fragment
645  82             <Properties> ?
646  83             <PropertyConstraints>
647  84               <PropertyConstraint property="xs:string"
648  85                                   constraintType="xs:anyURI"> +
649  86                 constraint ?
650  87               </PropertyConstraint>
651  88             </PropertyConstraints> ?
652  89           </Requirement>
653  90         </Requirements> ?
654  91
655  92         <Capabilities>
656  93           <Capability id="xs:ID" name="xs:string" type="xs:QName"> +
657  94             <Properties>
658  95               XML fragment
659  96             <Properties> ?
660  97             <PropertyConstraints>
661  98               <PropertyConstraint property="xs:string"
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
662 99                                    constraintType="xs:anyURI">
663 100                   constraint ?
664 101                 </PropertyConstraint> +
665 102               </PropertyConstraints> ?
666 103             </Capability>
667 104           </Capabilities> ?
668 105
669 106           <Policies>
670 107             <Policy name="xs:string"? policyType="xs:QName"
671 108                     policyRef="xs:QName"?>
672 109               policy specific content ?
673 110             </Policy> +
674 111           </Policies> ?
675 112
676 113           <DeploymentArtifacts>
677 114             <DeploymentArtifact name="xs:string" artifactType="xs:QName"
678 115                                 artifactRef="xs:QName"?>
679 116                artifact specific content ?
680 117             </DeploymentArtifact> +
681 118           </DeploymentArtifacts> ?
682 119         </NodeTemplate>
683 120       |
684 121         <RelationshipTemplate id="xs:ID" name="xs:string"?
685 122                               type="xs:QName">
686 123           <Properties>
687 124             XML fragment
688 125           </Properties> ?
689 126
690 127           <PropertyConstraints>
691 128             <PropertyConstraint property="xs:string"
692 129                                 constraintType="xs:anyURI">
693 130               constraint ?
694 131             </PropertyConstraint> +
695 132           </PropertyConstraints> ?
696 133
697 134           <SourceElement ref="xs:IDREF"/>
698 135           <TargetElement ref="xs:IDREF"/>
699 136
700 137           <RelationshipConstraints>
701 138             <RelationshipConstraint constraintType="xs:anyURI">
702 139               constraint ?
703 140             </RelationshipConstraint> +
704 141           </RelationshipConstraints> ?
705 142
706 143         </RelationshipTemplate>
707 144       ) +
708 145     </TopologyTemplate>
709 146
710 147     <Plans>
711 148       <Plan id="xs:ID"
712 149             name="xs:string"?
713 150             planType="xs:anyURI"
714 151             planLanguage="xs:anyURI">
715 152
716 153          <Precondition expressionLanguage="xs:anyURI">
717 154            condition
718 155          </Precondition> ?
719 156
```

 Copyright © OASIS Open 2013. All Rights Reserved.

720 721 ... (line numbers)

```
157          <InputParameters>
158            <InputParameter name="xs:string" type="xs:string"
159                            required="yes|no"?/> +
160          </InputParameters> ?
161
162          <OutputParameters>
163            <OutputParameter name="xs:string" type="xs:string"
164                             required="yes|no"?/> +
165          </OutputParameters> ?
166
167         (
168           <PlanModel>
169             actual plan
170           </PlanModel>
171          |
172           <PlanModelReference reference="xs:anyURI"/>
173         )
174
175       </Plan> +
176     </Plans> ?
177
178   </ServiceTemplate>
```

5.2 Properties

The `ServiceTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Service Template which MUST be unique within the target namespace.
- `name`: This OPTIONAL attribute specifies a descriptive name of the Service Template.
- `targetNamespace`: The value of this OPTIONAL attribute specifies the target namespace for the Service Template. If not specified, the Service Template will be added to the namespace declared by the `targetNamespace` attribute of the enclosing `Definitions` element.
- `substitutableNodeType`: This OPTIONAL attribute specifies a Node Type that can be substituted by this Service Template. If another Service Template contains a Node Template of the specified Node Type (or any Node Type this Node Type is derived from), this Node Template can be substituted by an instance of this Service Template that then provides the functionality of the substituted node. See section 3.5 for more details.
- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Service Template. Each tag is defined by a separate, nested `Tag` element.
  The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.
- `BoundaryDefinitions`: This OPTIONAL element specifies the properties the Service Template exposes beyond its boundaries, i.e. properties that can be observed from outside the Service Template. The `BoundaryDefinitions` element has the following properties.
  - `Properties`: This OPTIONAL element specifies global properties of the Service Template in the form of an XML fragment contained in the body of the `Properties` element. Those properties MAY be mapped to properties of components within the

 Copyright © OASIS Open 2013. All Rights Reserved.

Service Template to make them visible to the outside.
The `Properties` element has the following properties:

    - `PropertyMappings`: This OPTIONAL element specifies mappings of one or more of the Service Template's properties to properties of components within the Service Template (e.g. Node Templates, Relationship Templates, etc.). Each property mapping is defined by a separate, nested `PropertyMapping` element. The `PropertyMapping` element has the following properties:

        - `serviceTemplatePropertyRef`:This attribute identifies a property of the Service Template by means of an XPath expression to be evaluated on the XML fragment defining the Service Template's properties.

        - `targetObjectRef`: This attribute specifies the object that provides the property to which the respective Service Template property is mapped. The referenced target object MUST be one of Node Template, Requirement of a Node Template, Capability of a Node Template, or Relationship Template.

        - `targetPropertyRef`: This attribute identifies a property of the target object by means of an XPath expression to be evaluated on the XML fragment defining the target object's properties.

          Note: If a Service Template property is mapped to a property of a component within the Service Template, the XML schema type of the Service Template property and the mapped property MUST be compatible.

          Note: If a Service Template property is mapped to a property of a component within the Service Template, reading the Service Template property corresponds to reading the mapped property, and writing the Service Template property corresponds to writing the mapped property.

- `PropertyConstraints`: This OPTIONAL element specifies constraints on one or more of the Service Template's properties. Each constraint is specified by means of a separate, nested `PropertyConstraint` element.
  The `PropertyConstraint` element has the following properties:

    - `property`: This attribute identifies a property by means of an XPath expression to be evaluated on the XML fragment defining the Service Template's properties.

      Note: If the property affected by the property constraint is mapped to a property of a component within the Service Template, the property constraint SHOULD be compatible with any property constraint defined for the mapped property.

    - `constraintType`: This attribute specifies the type of constraint by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content.

    - The body of the `PropertyConstraint` element provides the actual constraint.
      Note: The body MAY be empty in case the `constraintType` URI already specifies the constraint appropriately. For example, a "read-only" constraint could be expressed solely by the `constraintType` URI.

- `Requirements`: This OPTIONAL element specifies Requirements exposed by the Service Template. Those Requirements correspond to Requirements of Node Templates within the Service Template that are propagated beyond the boundaries of the Service Template. Each Requirement is defined by a separate, nested `Requirement` element.
  The `Requirement` element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

821 - name: This OPTIONAL attribute allows for specifying a name of the Requirement
822 other than that specified by the referenced Requirement of a Node Template.
823 - ref: This attribute references a Requirement element of a Node Template
824 within the Service Template.

825 o Capabilities: This OPTIONAL element specifies Capabilities exposed by the
826 Service Template. Those Capabilities correspond to Capabilities of Node Templates
827 within the Service Template that are propagated beyond the boundaries of the Service
828 Template. Each Capability is defined by a separate, nested Capability element. The
829 Capability element has the following properties:
830 - name: This OPTIONAL attribute allows for specifying a name of the Capability
831 other than that specified by the referenced Capability of a Node Template.
832 - ref: This attribute references a Capability element of a Node Template
833 within the Service Template.

834 o Policies: This OPTIONAL element specifies global policies of the Service Template
835 related to a particular management aspect. All Policies defined within the Policies
836 element MUST be enforced by a TOSCA implementation, i.e. Policies are AND-
837 combined. Each policy is defined by a separate, nested Policy element.
838 The Policy element has the following properties:
839 - name: This OPTIONAL attribute allows for the definition of a name for the Policy.
840 If specified, this name MUST be unique within the containing Policies
841 element.
842 - policyType: This attribute specifies the type of this Policy. The QName value
843 of this attribute SHOULD correspond to the QName of a PolicyType defined
844 in the same Definitions document or in an imported document.
845
846 The policyType attribute specifies the artifact type specific content of the
847 Policy element body and indicates the type of Policy Template referenced by
848 the Policy via the policyRef attribute.
849 - policyRef: The QName value of this OPTIONAL attribute references a Policy
850 Template that is associated to the Service Template. This Policy Template can
851 be defined in the same TOSCA Definitions document, or it can be defined in a
852 separate document that is imported into the current Definitions document. The
853 type of Policy Template referenced by the policyRef attribute MUST be the
854 same type or a sub-type of the type specified in the policyType attribute.
855
856 Note: if no Policy Template is referenced, the policy specific content of the
857 Policy element alone is assumed to represent sufficient policy specific
858 information in the context of the Service Template.
859
860 Note: while Policy Templates provide invariant information about a non-functional
861 behavior (i.e. information that is context independent, such as the availability
862 class of an availability policy), the Policy element defined in a Service
863 Template can provide variant information (i.e. information that is context specific,
864 such as a specific heartbeat frequency for checking availability of a service) in
865 the policy specific body of the Policy element.

866 o Interfaces: This OPTIONAL element specifies the interfaces with operations that can
867 be invoked on complete service instances created from the Service Template.
868 The Interfaces element has the following properties:
869 - Interface: This element specifies one interfaces exposed by the Service
870 Template.
871 The Interface element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

- `name`: This attribute specifies the name of the interfaces as either a URI or an NCName that MUST be unique in the scope of the Service Template's boundary definitions.
- `Operation`: This element specifies one exposed operation of an interface exposed by the Service Template.

  An operation exposed by a Service Template maps to an internal component of the Service Template which actually provides the operation: it can be mapped to an operation provided by a Node Template (i.e. an operation defined by the Node Type specified in the `type` attribute of the Node Template), it can be mapped to an operation provided by a Relationship Template (i.e. an operation defined by the Relationship Type specified in the `type` attribute of the Relationship Template), or it can be mapped to a Plan of the Service Template.

  When an exposed operation is invoked on a service instance created from the Service Template, the operation or Plan mapped to the exposed operation will actually be invoked.

  The `Operation` element has the following properties:

  - `name`: This attribute specifies the name of the operation, which MUST be unique within the containing interface.
  - `NodeOperation`: This element specifies a reference to an operation of a Node Template.
    The `nodeRef` attribute of this element specifies a reference to the respective Node Template. The specific interface and operation to be mapped to the operation exposed by the Service Template are specified by means of the `interfaceName` and `operationName` attributes, respectively.

    Note: An interface and operation with the specified names MUST be defined by the Node Type (or one of its super types) defined in the `type` attribute of the referenced Node Template.
  - `RelationshipOperation`: This element specifies a reference to an operation of a Relationship Template.
    The `relationshipRef` attribute of this element specifies a reference to the respective Relationship Template. The specific interface and operation to be mapped to the operation exposed by the Service Template are specified by means of the `interfaceName` and `operationName` attributes, respectively.

    Note: An interface and operation with the specified names MUST be defined by the Relationship Type (or one of its super types) defined in the `type` attribute of the referenced Relationship Template.
  - `Plan`: This element specifies by means of its `planRef` attribute a reference to a Plan that provides the implementation of the operation exposed by the Service Template.

    One of `NodeOperation`, `RelationshipOperation` or `Plan` MUST be specified within the `Operation` element.

 Copyright © OASIS Open 2013. All Rights Reserved.

924 • `TopologyTemplate`: This element specifies the overall structure of the cloud application
925 defined by the Service Template, i.e. the components it consists of, and the relations between
926 those components. The components of a service are referred to as *Node Templates*, the relations
927 between the components are referred to as *Relationship Templates*.
928 The `TopologyTemplate` element has the following properties:

929 o `NodeTemplate`: This element specifies a kind of a component making up the cloud
930 application.
931 The `NodeTemplate` element has the following properties:

932 ▪ `id`: This attribute specifies the identifier of the Node Template. The identifier of
933 the Node Template MUST be unique within the target namespace.

934 ▪ `name`: This OPTIONAL attribute specifies the name of the Node Template.

935 ▪ `type`: The QName value of this attribute refers to the Node Type providing the
936 type of the Node Template.
937
938 Note: If the Node Type referenced by the `type` attribute of a Node Template is
939 declared as abstract, no instances of the specific Node Template can be created.
940 Instead, a substitution of the Node Template with one having a specialized,
941 derived Node Type has to be done at the latest during the instantiation time of
942 the Node Template.

943 ▪ `minInstances`: This integer attribute specifies the minimun number of
944 instances to be created when instantiating the Node Template. The default value
945 of this attribute is 1.The value of `minInstances` MUST NOT be less than 0.

946 ▪ `maxInstances`: This attribute specifies the maximum number of instances that
947 can be created when instantiating the Node Template. The default value of this
948 attribute is 1. If the string is set to "`unbounded`", an unbouded number of
949 instances can be created. The value of `maxInstances` MUST be 1 or greater
950 and MUST NOT be less than the value specified for `minInstances`.

951 ▪ `Properties`: Specifies initial values for one or more of the Node Type
952 Properties of the Node Type providing the property definitions in the concrete
953 context of the Node Template.
954 The initial values are specified by providing an instance document of the XML
955 schema of the corresponding Node Type Properties. This instance document
956 considers the inheritance structure deduced by the `DerivedFrom` property of
957 the Node Type referenced by the `type` attribute of the Node Template.
958 The instance document of the XML schema might not validate against the
959 existence constraints of the corresponding schema: not all Node Type properties
960 might have an initial value assigned, i.e. mandatory elements or attributes might
961 be missing in the instance provided by the `Properties` element. Once the
962 defined Node Template has been instantiated, any XML representation of the
963 Node Type properties MUST validate according to the associated XML schema
964 definition.

965 ▪ `PropertyConstraints`: Specifies constraints on the use of one or more of
966 the Node Type Properties of the Node Type providing the property definitions for
967 the Node Template. Each constraint is specified by means of a separate nested
968 `PropertyConstraint` element.
969 The `PropertyConstraint` element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

970 971 972 973
- `property`: The string value of this property is an XPath expression pointing to the property within the Node Type Properties document that is constrained within the context of the Node Template. More than one constraint MUST NOT be defined for each property.

974 975 976 977 978 979 980 981 982 983
- `constraintType`: The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content.

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

984 985 986 987 988
- `Requirements`: This element contains a list of requirements for the Node Template, according to the list of requirement definitions of the Node Type specified in the `type` attribute of the Node Template. Each requirement is specified in a separate nested `Requirement` element.
  The `Requirement` Element has the following properties:

989 990 991
  - `id`: This attribute specifies the identifier of the Requirement. The identifier of the Requirement MUST be unique within the target namespace.

992 993 994 995
  - `name`: This attribute specifies the name of the Requirement. The `name` and `type` of the Requirement MUST match the `name` and `type` of a Requirement Definition in the Node Type specified in the `type` attribute of the Node Template.

996 997 998
  - `type`: The QName value of this attribute refers to the Requirement Type definition of the Requirement. This Requirement Type denotes the semantics and well as potential properties of the Requirement.

999 1000 1001 1002 1003
  - `Properties`: This element specifies initial values for one or more of the Requirement Properties according to the Requirement Type providing the property definitions. Properties are provided in the form of an XML fragment. The same rules as outlined for the `Properties` element of the Node Template apply.

1004 1005 1006 1007 1008 1009
  - `PropertyConstraints`: This element specifies constraints on the use of one or more of the Properties of the Requirement Type providing the property definitions for the Requirement. Each constraint is specified by means of a separate nested `PropertyConstraint` element. The same rules as outlined for the `PropertyConstraints` element of the Node Template apply.

1010 1011 1012 1013 1014
- `Capabilities`: This element contains a list of capabilities for the Node Template, according to the list of capability definitions of the Node Type specified in the `type` attribute of the Node Template. Each capability is specified in a separate nested `Capability` element.
  The `Capability` Element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

- `id`: This attribute specifies the identifier of the Capability. The identifier of the Capability MUST be unique within the target namespace.
- `name`: This attribute specifies the name of the Capability. The `name` and `type` of the Capability MUST match the `name` and `type` of a Capability Definition in the Node Type specified in the `type` attribute of the Node Template.
- `type`: The QName value of this attribute refers to the Capability Type definition of the Capability. This Capability Type denotes the semantics and well as potential properties of the Capability.
- `Properties`: This element specifies initial values for one or more of the Capability Properties according to the Capability Type providing the property definitions. Properties are provided in the form of an XML fragment. The same rules as outlined for the `Properties` element of the Node Template apply.
- `PropertyConstraints`: This element specifies constraints on the use of one or more of the Properties of the Capability Type providing the property definitions for the Capability. Each constraint is specified by means of a separate nested `PropertyConstraint` element. The same rules as outlined for the `PropertyConstraints` element of the Node Template apply.

▪ `Policies`: This OPTIONAL element specifies policies associated with the Node Template. All Policies defined within the `Policies` element MUST be enforced by a TOSCA implementation, i.e. Policies are AND-combined. Each policy is specified by means of a separate nested `Policy` element.
The `Policy` element has the following properties:

- `name`: This OPTIONAL attribute allows for the definition of a name for the Policy. If specified, this name MUST be unique within the containing `Policies` element.
- `policyType`: This attribute specifies the type of this Policy. The QName value of this attribute SHOULD correspond to the QName of a `PolicyType` defined in the same Definitions document or in an imported document.

  The `policyType` attribute specifies the artifact type specific content of the `Policy` element body and indicates the type of Policy Template referenced by the Policy via the `policyRef` attribute.
- `policyRef`: The QName value of this OPTIONAL attribute references a Policy Template that is associated to the Node Template. This Policy Template can be defined in the same TOSCA Definitions document, or it can be defined in a separate document that is imported into the current Definitions document. The type of Policy Template referenced by the `policyRef` attribute MUST be the same type or a sub-type of the type specified in the `policyType` attribute.

  Note: if no Policy Template is referenced, the policy specific content of the `Policy` element alone is assumed to represent sufficient policy specific information in the context of the Node Template.

 Copyright © OASIS Open 2013. All Rights Reserved.

1062

1063 Note: while Policy Templates provide invariant information about a non-
1064 functional behavior (i.e. information that is context independent, such as
1065 the availability class of an availability policy), the `Policy` element
1066 defined in a Node Template can provide variant information (i.e.
1067 information that is context specific, such as a specific heartbeat
1068 frequency for checking availability of a component) in the policy specific
1069 body of the `Policy` element.

1070 ▪ `DeploymentArtifacts`: This element specifies the deployment artifacts
1071 relevant for the Node Template under definition. Its nested
1072 `DeploymentArtifact` elements specify details about individual deployment
1073 artifacts.
1074 The `DeploymentArtifact` element has the following properties:

1075 - `name`: This attribute specifies the name of the artifact. Uniqueness of the
1076 name within the scope of the encompassing Node Template SHOULD
1077 be guaranteed by the definition.

1078 - `artifactType`: This attribute specifies the type of this artifact. The
1079 QName value of this attribute SHOULD correspond to the QName of an
1080 `ArtifactType` defined in the same Definitions document or in an
1081 imported document.
1082
1083 The `artifactType` attribute specifies the artifact type specific content
1084 of the `DeploymentArtifact` element body and indicates the type of
1085 Artifact Template referenced by the Deployment Artifact via the
1086 `artifactRef` attribute.

1087 - `artifactRef`: This OPTIONAL attribute contains a QName that
1088 identifies an Artifact Template to be used as deployment artifact. This
1089 Artifact Template can be defined in the same Definitions document or in
1090 a separate, imported document.
1091 The type of Artifact Template referenced by the `artifactRef` attribute
1092 MUST be the same type or a sub-type of the type specified in the
1093 `artifactType` attribute.
1094
1095 Note: if no Artifact Template is referenced, the artifact type specific
1096 content of the `DeploymentArtifact` element alone is assumed to
1097 represent the actual artifact. For example, the contents of a simple config
1098 file could be defined in place within the `DeploymentArtifact`
1099 element.
1100
1101 Note, that a deployment artifact specified with the Node Template under
1102 definition overrides any deployment artifact of the same `name` and the
1103 same `artifactType` (or any Artifact Type it is derived from) specified
1104 with the Node Type Implementation implementing the Node Type given
1105 as value of the `type` attribute of the Node Template under definition.
1106 Otherwise, the deployment artifacts of Node Type Implementations and
1107 the deployment artifacts defined with the Node Template are combined.

1108 o `RelationshipTemplate`: This element specifies a kind of relationship between the
1109 components of the cloud application. For each specified Relationship Template the

 Copyright © OASIS Open 2013. All Rights Reserved.

source element and target element MUST be specified in the Topology Template.
The `RelationshipTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Relationship Template. The identifier of the Relationship Template MUST be unique within the target namespace.
- `name`: This OPTIONAL attribute specifies the name of the Relationship Template.
- `type`: The QName value of this property refers to the Relationship Type providing the type of the Relationship Template.

  Note: If the Relationship Type referenced by the `type` attribute of a Relationship Template is declared as abstract, no instances of the specific Relationship Template can be created. Instead, a substitution of the Relationship Template with one having a specialized, derived Relationship Type has to be done at the latest during the instantiation time of the Relationship Template.
- `Properties`: Specifies initial values for one or more of the Relationship Type Properties of the Relationship Type providing the property definitions in the concrete context of the Relationship Template.
  The initial values are specified by providing an instance document of the XML schema of the corresponding Relationship Type Properties. This instance document considers the inheritance structure deduced by the `DerivedFrom` property of the Relationship Type referenced by the `type` attribute of the Relationship Template.
  The instance document of the XML schema might not validate against the existence constraints of the corresponding schema: not all Relationship Type properties might have an initial value assigned, i.e. mandatory elements or attributes might be missing in the instance provided by the `Properties` element. Once the defined Relationship Template has been instantiated, any XML representation of the Relationship Type properties MUST validate according to the associated XML schema definition.
- `PropertyConstraints`: Specifies constraints on the use of one or more of the Relationship Type Properties of the Relationship Type providing the property definitions for the Relationship Template. Each constraint is specified by means of a separate nested `PropertyConstraint` element.
  The `PropertyConstraint` element has the following properties:
  - `property`: The string value of this property is an XPath expression pointing to the property within the Relationship Type Properties document that is constrained within the context of the Relationship Template. More than one constraint MUST NOT be defined for each property.
  - `constraintType`: The constraint type is specified by means of a URI, which defines both the semantic meaning of the constraint as well as the format of the content.

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

 Copyright © OASIS Open 2013. All Rights Reserved.

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

- `SourceElement`: This element specifies the origin of the relationship represented by the current Relationship Template.
  The `SourceElement` element has the following property:
  - `ref`: This attribute references by ID a Node Template or a Requirement of a Node Template within the same Service Template document that is the source of the Relationship Template.

    If the Relationship Type referenced by the `type` attribute defines a constraint on the valid source of the relationship by means of its `ValidSource` element, the `ref` attribute of `SourceElement` MUST reference an object the type of which complies with the valid source constraint of the respective Relationship Type.

    In the case where a Node Type is defined as valid source in the Relationship Type definition, the `ref` attribute MUST reference a Node Template of the corresponding Node Type (or of a sub-type).

    In the case where a Requirement Type is defined a valid source in the Relationship Type definition, the `ref` attribute MUST reference a Requirement of the corresponding Requirement Type within a Node Template.
- `TargetElement`: This element specifies the target of the relationship represented by the current Relationship Template.
  The TargetElement element has the following property:
  - `ref`: This attribute references by ID a Node Template or a Capability of a Node Template within the same Service Template document that is the target of the Relationship Template.

    If the Relationship Type referenced by the `type` attribute defines a constraint on the valid source of the relationship by means of its `ValidTarget` element, the `ref` attribute of `TargetElement` MUST reference an object the type of which complies with the valid source constraint of the respective Relationship Type.

    In case a Node Type is defined as valid target in the Relationship Type definition, the `ref` attribute MUST reference a Node Template of the corresponding Node Type (or of a sub-type).

    In case a Capability Type is defined a valid target in the Relationship Type definition, the `ref` attribute MUST reference a Capability of the corresponding Capability Type within a Node Template.
- `RelationshipConstraints`: This element specifies a list of constraints on the use of the relationship in separate nested `RelationshipConstraint` elements.
  The `RelationshipConstraint` element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

1205 - `constraintType`: This attribute specifies the type of relationship constraint by means of a URI. Depending on the type, the body of the `RelationshipConstraint` element might contain type specific content that further details the actual constraint.

1209 - `Plans`: This element specifies the operational behavior of the service. A `Plan` contained in the `Plans` element can specify how to create, terminate or manage the service.
  The `Plan` element has 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.
  - `planLanguage`: This attribute denotes the process modeling language (or metamodel) used to specify the plan. For example, “http://www.omg.org/spec/BPMN/20100524/MODEL” would specify that BPMN 2.0 has been used to model the plan.

    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.
  - `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.
  - `InputParameters`: This OPTIONAL property contains a list of one or more input parameter definitions for the Plan, each defined in a nested, separate `InputParameter` element.
    The `InputParameter` element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

- `name`: This attribute specifies the name of the input parameter, which MUST be unique within the set of input parameters defined for the operation.
- `type`: This attribute specifies the type of the input parameter.
- `required`: This OPTIONAL attribute specifies whether or not the input parameter is REQUIRED (`required` attribute with a value of “yes” – default) or OPTIONAL (`required` attribute with a value of “no”).

- `OutputParameters`: This OPTIONAL property contains a list of one or more output parameter definitions for the Plan, each defined in a nested, separate `OutputParameter` element.
  The `OutputParameter` element has the following properties:
  - `name`: This attribute specifies the name of the output parameter, which MUST be unique within the set of output parameters defined for the operation.
  - `type`: This attribute specifies the type of the output parameter.
  - `required`: This OPTIONAL attribute specifies whether or not the output parameter is REQUIRED (`required` attribute with a value of “yes” – default) or OPTIONAL (`required` attribute with a value of “no”).
- `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.

5.3 Example

The following Service Template defines a Topology Template containing two Node Templates called “MyApplication” and “MyAppServer”. These Node Templates have the node types “Application” and “ApplicationServer”. The Node Template “MyApplication” is instantiated exactly once. Two of its Node Type Properties are initialized by a corresponding `Properties` element. The Node Template “MyAppServer” can be instantiated as many times as needed. The “MyApplication” Node Template is connected with the “MyAppServer” Node Template via the Relationship Template named “MyHostedRelationship”; the behavior and semantics of the Relationship Template is defined in the Relationship Type “HostedOn”, saying that “MyApplication” is hosted on “MyAppServer”. The Service Template further defines a Plan “UpdateApplication” for performing an update of the “MyApplication” application hosted on the application server. This Plan refers to a BPMN 2.0 process definition contained in a separate file.

```
01 <ServiceTemplate id="MyService"
02                  name="My Service">
03
04   <TopologyTemplate>
05
06     <NodeTemplate id="MyApplication"
07                   name="My Application"
08                   type="my:Application">
09       <Properties>
10         <ApplicationProperties>
11           <Owner>Frank</Owner>
12           <InstanceName>Thomas’ favorite application</InstanceName>
13         </ApplicationProperties>
14        </Properties>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
1298  15      </NodeTemplate>
1299  16
1300  17      <NodeTemplate id="MyAppServer"
1301  18                    name="My Application Server"
1302  19                    type="my:ApplicationServer"
1303  20                    minInstances="0"
1304  21                    maxInstances="unbounded"/>
1305  22
1306  23      <RelationshipTemplate id="MyDeploymentRelationship"
1307  24                            type="my:deployedOn">
1308  25        <SourceElement ref="MyApplication"/>
1309  26        <TargetElement ref="MyAppServer"/>
1310  27      </RelationshipTemplate>
1311  28
1312  29    </TopologyTemplate>
1313  30
1314  31    <Plans>
1315  32      <Plan id="UpdateApplication"
1316  33          planType="http://www.example.com/UpdatePlan"
1317  34          planLanguage="http://www.omg.org/spec/BPMN/20100524/MODEL">
1318  35        <PlanModelReference reference="plans:UpdateApp"/>
1319  36      </Plan>
1320  37    </Plans>
1321  38
1322  39  </ServiceTemplate>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

6 Node Types

This chapter specifies how *Node Types* are defined. A Node Type is a reusable entity that defines the type of one or more Node Templates. As such, a Node Type defines the structure of observable properties via a *Properties Definition*, i.e. the names, data types and allowed values the properties defined in Node Templates using a Node Type or instances of such Node Templates can have.

A Node Type can inherit properties from another Node Type by means of the `DerivedFrom` element. Node Types might be declared as abstract, meaning that they cannot be instantiated. The purpose of such abstract Node Types is to provide common properties and behavior for re-use in specialized, derived Node Types. Node Types might also be declared as final, meaning that they cannot be derived by other Node Types.

A Node Type can declare to expose certain requirements and capabilities (see section 3.4) by means of `RequirementDefinition` elements or `CapabilityDefinition` elements, respectively.

The functions that can be performed on (an instance of) a corresponding Node Template are defined by the *Interfaces* of the Node Type. Finally, management Policies are defined for a Node Type.

6.1 XML Syntax

The following pseudo schema defines the XML syntax of Node Types:

```
01 <NodeType name="xs:NCName" targetNamespace="xs:anyURI"?
02           abstract="yes|no"? final="yes|no"?>
03
04   <Tags>
05     <Tag name="xs:string" value="xs:string"/> +
06   </Tags> ?
07
08   <DerivedFrom typeRef="xs:QName"/> ?
09
10   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
11
12   <RequirementDefinitions>
13     <RequirementDefinition name="xs:string"
14                            requirementType="xs:QName"
15                            lowerBound="xs:integer"?
16                            upperBound="xs:integer | xs:string"?>
17       <Constraints>
18         <Constraint constraintType="xs:anyURI">
19           constraint type specific content
20         </Constraint> +
21       </Constraints> ?
22     </RequirementDefinition> +
23   </RequirementDefinitions> ?
24
25   <CapabilityDefinitions>
26     <CapabilityDefinition name="xs:string"
27                           capabilityType="xs:QName"
28                           lowerBound="xs:integer"?
29                           upperBound="xs:integer | xs:string"?>
30       <Constraints>
31         <Constraint constraintType="xs:anyURI">
32           constraint type specific content
33         </Constraint> +
34       </Constraints> ?
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
35     </CapabilityDefinition> +
36   </CapabilityDefinitions>
37
38   <InstanceStates>
39     <InstanceState state="xs:anyURI"> +
40   </InstanceStates> ?
41
42   <Interfaces>
43     <Interface name="xs:NCName | xs:anyURI">
44       <Operation name="xs:NCName">
45         <InputParameters>
46           <InputParameter name="xs:string" type="xs:string"
47                           required="yes|no"?/> +
48         </InputParameters> ?
49         <OutputParameters>
50           <OutputParameter name="xs:string" type="xs:string"
51                            required="yes|no"?/> +
52         </OutputParameters> ?
53       </Operation> +
54     </Interface> +
55   </Interfaces> ?
56
57 </NodeType>
```

6.2 Properties

The `NodeType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Node Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Node Type will be added. If not specified, the Node Type definition will be added to the target namespace of the enclosing Definitions document.
- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Node Templates that use this Node Type as their type. If a Node Type includes a Requirement Definition or Capability Definition of an abstract Requirement Type or Capability Type, respectively, the Node Type MUST be declared as abstract as well.

  As a consequence, the corresponding abstract Node Type referenced by any Node Template has to be substituted by a Node Type derived from the abstract Node Type at the latest during the instantiation time of a Node Template.

  Note: an abstract Node Type MUST NOT be declared as final.
- `final`: This OPTIONAL attribute specifies that other Node Types MUST NOT be derived from this Node Type.

  Note: a final Node Type MUST NOT be declared as abstract.
- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Node Type. Each tag is defined by a separate, nested `Tag` element. The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.

Copyright © OASIS Open 2013. All Rights Reserved.

1424 • `DerivedFrom`: This is an OPTIONAL reference to another Node Type from which this Node
1425 Type derives. Conflicting definitions are resolved by the rule that local new definitions always
1426 override derived definitions. See section 6.3 Derivation Rules for details.
1427 The `DerivedFrom` element has the following properties:

1428 - `typeRef`: The QName specifies the Node Type from which this Node Type derives its
1429 definitions.

1430 • `PropertiesDefinition`: This element specifies the structure of the observable properties
1431 of the Node Type, such as its configuration and state, by means of XML schema.
1432 The `PropertiesDefinition` element has one but not both of the following properties:

1433 - `element`: This attribute provides the QName of an XML element defining the structure
1434 of the Node Type Properties.

1435 - `type`: This attribute provides the QName of an XML (complex) type defining the
1436 structure of the Node Type Properties.

1437 • `RequirementDefinitions`: This OPTIONAL element specifies the requirements that the
1438 Node Type exposes (see section 3.4 for details). Each requirement is defined in a nested
1439 `RequirementDefinition` element.
1440 The `RequirementDefinition` element has the following properties:

1441 - `name`: This attribute specifies the name of the defined requirement and MUST be unique
1442 within the `RequirementsDefinitions` of the current Node Type.
1443
1444 Note that one Node Type might define multiple requirements of the same Requirement
1445 Type, in which case each occurrence of a requirement definition is uniquely identified by
1446 its name. For example, a Node Type for an application might define two requirements for
1447 a database (i.e. of the same Requirement Type) where one could be named
1448 "customerDatabase" and the other one could be named "productsDatabase".

1449 - `requirementType`: This attribute identifies by QName the Requirement Type that is
1450 being defined by the current `RequirementDefinition`.

1451 - `lowerBound`: This OPTIONAL attribute specifies the lower boundary by which a
1452 requirement MUST be matched for Node Templates according to the current Node Type,
1453 or for instances created for those Node Templates. The default value for this attribute is
1454 one. A value of zero would indicate that matching of the requirement is optional.

1455 - `upperBound`: This OPTIONAL attribute specifies the upper boundary by which a
1456 requirement MUST be matched for Node Templates according to the current Node Type,
1457 or for instances created for those Node Templates. The default value for this attribute is
1458 one. A value of "`unbounded`" indicates that there is no upper boundary.
1459 `Constraints`: This OPTIONAL element contains a list of `Constraint` elements that
1460 specify additional constraints on the requirement definition. For example, if a database is
1461 needed a constraint on supported SQL features might be expressed.
1462 The nested `Constraint` element has the following properties:

1463 - `constraintType`: This attribute specifies the type of constraint. According to
1464 this type, the body of the `Constraint` element will contain type specific
1465 content.

1466 • `CapabilityDefinitions`: This OPTIONAL element specifies the capabilities that the Node
1467 Type exposes (see section 3.4 for details). Each capability is defined in a nested
1468 `CapabilityDefinition` element.
1469 The `CapabilityDefinition` element has the following properties:

1470 - `name`: This attribute specifies the name of the defined capability and MUST be unique
1471 within the `CapabilityDefinitions` of the current Node Type.
1472

 Copyright © OASIS Open 2013. All Rights Reserved.

Note that one Node Type might define multiple capabilities of the same Capability Type, in which case each occurrence of a capability definition is uniquely identified by its name.

- capabilityType: This attribute identifies by QName the Capability Type of capability that is being defined by the current CapabilityDefinition.
- lowerBound: This OPTIONAL attribute specifies the lower boundary of requiring nodes that the defined capability can serve. The default value for this attribute is one. A value of zero is invalid, since this would mean that the capability cannot actually satisfy any requiring nodes.
- upperBound: This OPTIONAL attribute specifies the upper boundary of client requirements the defined capability can serve. The default value for this attribute is one. A value of "unbounded" indicates that there is no upper boundary.
- Constraints: This OPTIONAL element contains a list of Constraint elements that specify additional constraints on the capability definition.
  The nested Constraint element has the following properties:
  - constraintType: This attribute specifies the type of constraint. According to this type, the body of the Constraint element will contain type specific content.

- InstanceStates: This OPTIONAL element lists the set of states an instance of this Node Type can occupy. Those states are defined in nested InstanceState elements.
  The InstanceState element has the following nested properties:
  - state: This attribute specifies a URI that identifies a potential state.
- Interfaces: This element contains the definitions of the operations that can be performed on (instances of) this Node Type. Such operation definitions are given in the form of nested Interface elements.
  The Interface element has the following properties:
  - name: The name of the interface. This name is either a URI or it is an NCName that MUST be unique in the scope of the Node Type being defined.
  - Operation: This element defines an operation available to manage particular aspects of the Node Type.

    The Operation element has the following properties:
    - name: This attribute defines the name of the operation and MUST be unique within the containing Interface of the Node Type.
    - InputParameters: This OPTIONAL property contains a list of one or more input parameter definitions, each defined in a nested, separate InputParameter element.
      The InputParameter element has the following properties:
      - name: This attribute specifies the name of the input parameter, which MUST be unique within the set of input parameters defined for the operation.
      - type: This attribute specifies the type of the input parameter.
      - required: This OPTIONAL attribute specifies whether or not the input parameter is REQUIRED (required attribute with a value of "yes" – default) or OPTIONAL (required attribute with a value of "no").
    - OutputParameters: This OPTIONAL property contains a list of one or more output parameter definitions, each defined in a nested, separate OutputParameter element.
      The OutputParameter element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

1521 1522 1523
- `name`: This attribute specifies the name of the output parameter, which MUST be unique within the set of output parameters defined for the operation.
- `type`: This attribute specifies the type of the output parameter.
- `required`: This OPTIONAL attribute specifies whether or not the output parameter is REQUIRED (`required` attribute with a value of "yes" – default) or OPTIONAL (`required` attribute with a value of "no").

6.3 Derivation Rules

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

- Node Type Properties: It is assumed that the XML element (or type) representing the Node Type Properties extends the XML element (or type) of the Node Type Properties of the Node Type referenced in the `DerivedFrom` element.
- Requirements and capabilities: The set of requirements or capabilities of the Node Type under definition consists of the set union of requirements or capabilities defined by the Node Type derived from and the requirements or capabilities defined by the Node Type under definition.

  In cases where the Node Type under definition defines a requirement or capability with a certain name where the Node Type derived from already contains a respective definition with the same name, the definition in the Node Type under definition overrides the definition of the Node Type derived from. In such a case, the requirement definition or capability definition, respectively, MUST reference a Requirement Type or Capability Type that is derived from the one in the corresponding requirement definition or capability definition of the Node Type derived from.
- Instance States: The set of instance states of the Node Type under definition consists of the set union of the instances states defined by the Nodes Type derived from and the instance states defined by the Node Type under definition. A set of instance states of the same name will be combined into a single instance state of the same name.
- Interfaces: The set of interfaces of the Node Type under definition consists of the set union of interfaces defined by the Node Type derived from and the interfaces defined by the Node Type under definition.
  Two interfaces of the same name will be combined into a single, derived interface with the same name. The set of operations of the derived interface consists of the set union of operations defined by both interfaces. An operation defined by the Node Type under definition substitutes an operation with the same name of the Node Type derived from.

6.4 Example

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

```
01 <Definitions id="MyDefinitions" name="My Definitions"
02              targetNamespace="http://www.example.com/sample">
03
04   <NodeType name="Project">
05
06     <documentation xml:lang="EN">
07       A reusable definition of a node type supporting
08       the creation of new projects.
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
1567 09     </documentation>
1568 10
1569 11     <PropertiesDefinition element="ProjectProperties"/>
1570 12
1571 13     <InstanceStates>
1572 14       <InstanceState state="www.example.com/active"/>
1573 15       <InstanceState state="www.example.com/onHold"/>
1574 16     </InstanceStates>
1575 17
1576 18     <Interfaces>
1577 19       <Interface name="ProjectInterface">
1578 20         <Operation name="CreateProject">
1579 21           <InputParameters>
1580 22             <InputParamter name="ProjectName"
1581 23                            type="xs:string"/>
1582 24             <InputParamter name="Owner"
1583 25                            type="xs:string"/>
1584 26             <InputParamter name="AccountID"
1585 27                            type="xs:string"/>
1586 28           </InputParameters>
1587 29         </Operation>
1588 30       </Interface>
1589 31     </Interfaces>
1590 32   </NodeType>
1591 33
1592 34 </Definitions>
```

1593 The Node Type “Project” has three Node Type Properties defined as an XML elelment in the `Types`
1594 element definition of the Service Template document: Owner, ProjectName and AccountID which are all
1595 of type “xs:string”. An instance of the Node Type “Project” could be “active” (more precise in state
1596 www.example.com/active) or “on hold” (more precise in state “www.example.com/onHold”). A single
1597 Interface is defined for this Node Type, and this Interface is defined by an Operation, i.e. its actual
1598 implementation is defined by the definition of the Operation. The Operation has the name CreateProject
1599 and three Input Parameters (exploiting the default value “yes” of the attribute `required` of the
1600 `InputParameter` element). The names of these Input Parameters are ProjectName, Owner and
1601 AccountID, all of type “xs:string”.

 Copyright © OASIS Open 2013. All Rights Reserved.

7 Node Type Implementations

This chapter specifies how *Node Type Implementations* are defined. A Node Type Implementation represents the executable code that implements a specific Node Type. It provides a collection of executables implementing the interface operations of a Node Type (aka implementation artifacts) and the executables needed to materialize instances of Node Templates referring to a particular Node Type (aka deployment artifacts). The respective executables are defined as separate Artifact Templates and are referenced from the implementation artifacts and deployment artifacts of a Node Type Implementation.

While Artifact Templates provide invariant information about an artifact – i.e. information that is context independent like the file name of the artifact – implementation or deployment artifacts can provide variant (or context specific) information, such as authentication data or deployment paths for a specific environment.

Node Type Implementations can specify hints for a TOSCA container that enable proper selection of an implementation that fits into a particular environment by means of Required Container Features definitions.

7.1 XML Syntax

The following pseudo schema defines the XML syntax of Node Type Implementations:

```
01 <NodeTypeImplementation name="xs:NCName" targetNamespace="xs:anyURI"?
02                           nodeType="xs:QName"
03                           abstract="yes|no"?
04                           final="yes|no"?>
05
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <DerivedFrom nodeTypeImplementationRef="xs:QName"/> ?
11
12   <RequiredContainerFeatures>
13     <RequiredContainerFeature feature="xs:anyURI"/> +
14   </RequiredContainerFeatures> ?
15
16   <ImplementationArtifacts>
17     <ImplementationArtifact interfaceName="xs:NCName | xs:anyURI"?
18                             operationName="xs:NCName"?
19                             artifactType="xs:QName"
20                             artifactRef="xs:QName"?>
21         artifact specific content ?
22     <ImplementationArtifact> +
23   </ImplementationArtifacts> ?
24
25   <DeploymentArtifacts>
26     <DeploymentArtifact name="xs:string" artifactType="xs:QName"
27                         artifactRef="xs:QName"?>
28         artifact specific content ?
29     <DeploymentArtifact> +
30   </DeploymentArtifacts> ?
31
32 </NodeTypeImplementation>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

1650 ## 7.2 Properties

1651 The `NodeTypeImplementation` element has the following properties:

1652 • `name`: This attribute specifies the name or identifier of the Node Type Implementation, which
1653 MUST be unique within the target namespace.

1654 • `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the
1655 definition of the Node Type Implementation will be added. If not specified, the Node Type
1656 Implementation will be added to the target namespace of the enclosing Definitions document.

1657 • `nodeType`: The QName value of this attribute specifies the Node Type implemented by this
1658 Node Type Implementation.

1659 • `abstract`: This OPTIONAL attribute specifies that this Node Type Implementation cannot be
1660 used directly as an implementation for the Node Type specified in the `nodeType` attribute.
1661
1662 For example, a Node Type implementer might decide to deliver only part of the implementation of
1663 a specific Node Type (i.e. for only some operations) for re-use purposes and require the
1664 implementation for specific operations to be delivered in a more concrete, derived Node Type
1665 Implementation.
1666
1667 Note: an abstract Node Type Implementation MUST NOT be declared as final.

1668 • `final`: This OPTIONAL attribute specifies that other Node Type Implementations MUST NOT
1669 be derived from this Node Type Implementation.
1670
1671 Note: a final Node Type Implementation MUST NOT be declared as abstract.

1672 • `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by
1673 the author to describe the Node Type Implementation. Each tag is defined by a separate, nested
1674 `Tag` element.
1675 The `Tag` element has the following properties:

1676 o `name`: This attribute specifies the name of the tag.

1677 o `value`: This attribute specifies the value of the tag.
1678
1679 Note: The name/value pairs defined in tags have no normative interpretation.

1680 • `DerivedFrom`: This is an OPTIONAL reference to another Node Type Implementation from
1681 which this Node Type Implementation derives. See section 7.3 Derivation Rules for details.
1682 The `DerivedFrom` element has the following properties:

1683 o `nodeTypeImplementationRef`: The QName specifies the Node Type
1684 Implementation from which this Node Type Implementation derives.

1685 • `RequiredContainerFeatures`: An implementation of a Node Type might depend on
1686 certain features of the environment it is executed in, such as specific (potentially proprietary) APIs
1687 of the TOSCA container. For example, an implementation to deploy a virtual machine based on
1688 an image could require access to some API provided by a public cloud, while another
1689 implementation could require an API of a vendor-specific virtual image library. Thus, the contents
1690 of the `RequiredContainerFeatures` element provide "hints" to the TOSCA container
1691 allowing it to select the appropriate Node Type Implementation if multiple alternatives are
1692 provided.
1693 Each such dependency is defined by a separate `RequiredContainerFeature` element.
1694 The `RequiredContainerFeature` element has the following properties:

1695 o `feature`: The value of this attribute is a URI that denotes the corresponding needed
1696 feature of the environment.

 Copyright © OASIS Open 2013. All Rights Reserved.

1697 • `ImplementationArtifacts`: This element specifies a set of implementation artifacts for
1698 interfaces or operations of a Node Type.
1699 The `ImplementationArtifacts` element has the following properties:

1700 o `ImplementationArtifact`: This element specifies one implementation artifact of
1701 an interface or an operation.
1702
1703 Note: Multiple implementation artifacts might be needed to implement a Node Type
1704 according to the attributes defined below. An implementation artifact MAY serve as
1705 implementation for all interfaces and all operations defined for the Node Type, it MAY
1706 serve as implementation for one interface (and all its operations), or it MAY serve as
1707 implementation for only one specific operation.
1708
1709 The `ImplementationArtifact` element has the following properties:

1710 ▪ `name`: This attribute specifies the name of the artifact, which SHOULD be unique
1711 within the scope of the encompassing Node Type Implementation.

1712 ▪ `interfaceName`: This OPTIONAL attribute specifies the name of the interface
1713 that is implemented by the actual implementation artifact. If not specified, the
1714 implementation artifact is assumed to provide the implementation for all
1715 interfaces defined by the Node Type referred to by the `nodeType` attribute of
1716 the containing `NodeTypeImplementation`.

1717 ▪ `operationName`: This OPTIONAL attribute specifies the name of the
1718 operation that is implemented by the actual implementation artifact. If specified,
1719 the `interfaceName` MUST be specified and the specified `operationName`
1720 MUST refer to an operation of the specified interface. If not specified, the
1721 implementation artifact is assumed to provide the implementation for all
1722 operations defined within the specified interface.

1723 ▪ `artifactType`: This attribute specifies the type of this artifact. The QName
1724 value of this attribute SHOULD correspond to the QName of an
1725 `ArtifactType` defined in the same Definitions document or in an imported
1726 document.
1727
1728 The `artifactType` attribute specifies the artifact type specific content of the
1729 `ImplementationArtifact` element body and indicates the type of Artifact
1730 Template referenced by the Implementation Artifact via the `artifactRef`
1731 attribute.

1732 ▪ `artifactRef`: This OPTIONAL attribute contains a QName that identifies an
1733 Artifact Template to be used as implementation artifact. This Artifact Template
1734 can be defined in the same Definitions document or in a separate, imported
1735 document.
1736 The type of Artifact Template referenced by the `artifactRef` attribute MUST
1737 be the same type or a sub-type of the type specified in the `artifactType`
1738 attribute.
1739
1740 Note: if no Artifact Template is referenced, the artifact type specific content of the
1741 `ImplementationArtifact` element alone is assumed to represent the
1742 actual artifact. For example, a simple script could be defined in place within the
1743 `ImplementationArtifact` element.

1744 • `DeploymentArtifacts`: This element specifies a set of deployment artifacts relevant for
1745 materializing instances of nodes of the Node Type being implemented.
1746 The `DeploymentArtifacts` element has the following properties:

1747 o `DeploymentArtifact`: This element specifies one deployment artifact.
1748

 Copyright © OASIS Open 2013. All Rights Reserved.

Note: Multiple deployment artifacts MAY be defined in a Node Type Implementation. One reason could be that multiple artifacts (maybe of different types) are needed to materialize a node as a whole. Another reason could be that alternative artifacts are provided for use in different contexts (e.g. different installables of a software for use in different operating systems).

The `DeploymentArtifact` element has the following properties:

- `name`: This attribute specifies the name of the artifact, which SHOULD be unique within the scope of the encompassing Node Type Implementation.
- `artifactType`: This attribute specifies the type of this artifact. The QName value of this attribute SHOULD correspond to the QName of an `ArtifactType` defined in the same Definitions document or in an imported document.

  The `artifactType` attribute specifies the artifact type specific content of the `DeploymentArtifact` element body and indicates the type of Artifact Template referenced by the Deployment Artifact via the `artifactRef` attribute.
- `artifactRef`: This OPTIONAL attribute contains a QName that identifies an Artifact Template to be used as deployment artifact. This Artifact Template can be defined in the same Definitions document or in a separate, imported document.
  The type of Artifact Template referenced by the `artifactRef` attribute MUST be the same type or a sub-type of the type specified in the `artifactType` attribute.

  Note: if no Artifact Template is referenced, the artifact type specific content of the `DeploymentArtifact` element alone is assumed to represent the actual artifact. For example, the contents of a simple config file could be defined in place within the `DeploymentArtifact` element.

7.3 Derivation Rules

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

- Implementation Artifacts: The set of implementation artifacts of a Node Type Implementation consists of the set union of implementation artifacts defined by the Node Type Implementation itself and the implementation artifacts defined by any Node Type Implemenation the Node Type Implementation is derived from.
  An implementation artifact defined by a Node Type Implementation overrides an implementation artifact having the same interface name and operation name of a Node Type Implementation the Node Type Implementation is derived from.
  If an implementation artifact defined in a Node Type Implementation specifies only an interface name, it substitutes implementation artifacts having the same interface name (with or without an operation name defined) of any Node Type Implementation the Node Type Implementation is derived from. In this case, the implementation of a complete interface of a Node Type is overridden.
  If an implementation artifact defined in a Node Type Implementation neither defines an interface name nor an operation name, it overrides all implementation artifacts of any Node Type Implementation the Node Type Implementation is derived from. In this case, the complete implementation of a Node Type is overridden.

 Copyright © OASIS Open 2013. All Rights Reserved.

1797 - Deployment Artifacts: The set of deployment artifacts of a Node Type Implementation consists of
1798 the set union of the deployment artifacts defined by the Nodes Type Implementation itself and the
1799 deployment artifacts defined by any Node Type Implementation the Node Type Implementation is
1800 derived from. A deployment artifact defined by a Node Type Implementation overrides a
1801 deployment artifact with the same name and type (or any type it is derived from) of any Node
1802 Type Implementation the Node Type Implementation is derived from.

1803 ## 7.4 Example

1804 The following example defines the Node Type Implementation “MyDBMSImplementation”. This is an
1805 implementation of a Node Type “DBMS”.

```
1806 01 <Definitions id="MyImpls" name="My Implementations"
1807 02   targetNamespace="http://www.example.com/SampleImplementations"
1808 03   xmlns:bn="http://www.example.com/BaseNodeTypes"
1809 04   xmlns:ba="http://www.example.com/BaseArtifactTypes"
1810 05   xmlns:sa="http://www.example.com/SampleArtifacts">
1811 06
1812 07   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
1813 08           namespace="http://www.example.com/BaseArtifactTypes"/>
1814 09
1815 10   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
1816 11           namespace="http://www.example.com/BaseNodeTypes"/>
1817 12
1818 13   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
1819 14           namespace="http://www.example.com/SampleArtifacts"/>
1820 15
1821 16   <NodeTypeImplementation name="MyDBMSImplementation"
1822 17                           nodeType="bn:DBMS">
1823 18
1824 19     <ImplementationArtifacts>
1825 20       <ImplementationArtifact interfaceName="MgmtInterface"
1826 21                               artifactType="ba:WARFile"
1827 22                               artifactRef="sa:MyMgmtWebApp">
1828 23       </ImplementationArtifact>
1829 24     </ImplementationArtifacts>
1830 25
1831 26     <DeploymentArtifacts>
1832 27       <DeploymentArtifact name="MyDBMS"
1833 28                           artifactType="ba:ZipFile"
1834 29                           artifactRef="sa:MyInstallable">
1835 30       </DeploymentArtifact>
1836 31     </DeploymentArtifacts>
1837 32
1838 33   </NodeTypeImplementation>
1839 34
1840 35 </Definitions>
```

1841 The Node Type Implementation contains the “MyDBMSManagement” implementation artifact, which is an
1842 artifact for the “MgmtInterface” Interface that has been defined for the “DBMS” base Node Type. The type
1843 of this artifact is a “WARFile” that has been defined as base Artifact Type. The implementation artifact
1844 refers to the “MyMgmtWebApp” Artifact Template that has been defined before.

1845 The Node Type Implementation further contains the “MyDBMS” deployment artifact, which is a software
1846 installable used for instantiating the “DBMS” Node Type. This software installable is a “ZipFile” that has
1847 been separately defined as the “MyInstallable” Artifact Template before.

 Copyright © OASIS Open 2013. All Rights Reserved.

8 Relationship Types

This chapter specifies how *Relationship Types* are defined. A Relationship Type is a reusable entity that defines the type of one or more Relationship Templates between Node Templates. As such, a Relationship Type can define the structure of observable properties via a *Properties Definition*, i.e. the names, data types and allowed values the properties defined in Relationship Templates using a Relationship Type or instances of such Relationship Templates can have.

The operations that can be performed on (an instance of) a corresponding Relationship Template are defined by the *Interfaces* of the Relationship Type. Furthermore, a Relationship Type defines the potential states an instance of it might reveal at runtime.

A Relationship Type can inherit the definitions defined in another Relationship Type by means of the `DerivedFrom` element. Relationship Types might be declared as abstract, meaning that they cannot be instantiated. The purpose of such abstract Relationship Types is to provide common properties and behavior for re-use in specialized, derived Relationship Types. Relationship Types might also be declared as final, meaning that they cannot be derived by other Relationship Types.

8.1 XML Syntax

The following pseudo schema defines the XML syntax of Relationship Types:

```
01 <RelationshipType name="xs:NCName"
02                   targetNamespace="xs:anyURI"?
03                   abstract="yes|no"?
04                   final="yes|no"?> +
05
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <DerivedFrom typeRef="xs:QName"/> ?
11
12   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
13
14   <InstanceStates>
15     <InstanceState state="xs:anyURI"> +
16   </InstanceStates> ?
17
18   <SourceInterfaces>
19     <Interface name="xs:NCName | xs:anyURI">
20       ...
21     </Interface> +
22   </SourceInterfaces> ?
23
24   <TargetInterfaces>
25     <Interface name="xs:NCName | xs:anyURI">
26       ...
27     </Interface> +
28   </TargetInterfaces> ?
29
30   <ValidSource typeRef="xs:QName"/> ?
31
32   <ValidTarget typeRef="xs:QName"/> ?
33
34 </RelationshipType>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

8.2 Properties

The `RelationshipType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Relationship Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Relationship Type will be added. If not specified, the Relationship Type definition will be added to the target namespace of the enclosing Definitions document.
- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Relationship Templates that use this Relationship Type as their type.

  As a consequence, the corresponding abstract Relationship Type referenced by any Relationship Template has to be substituted by a Relationship Type derived from the abstract Relationship Type at the latest during the instantiation time of a Relationship Template.

  Note: an abstract Relationship Type MUST NOT be declared as final.
- `final`: This OPTIONAL attribute specifies that other Relationship Types MUST NOT be derived from this Relationship Type.

  Note: a final Relationship Type MUST NOT be declared as abstract.
- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Relationship Type. Each tag is defined by a separate, nested `Tag` element.
  The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.
- `DerivedFrom`: This is an OPTIONAL reference to another Relationship Type from which this Relationship Type is derived. Conflicting definitions are resolved by the rule that local new definitions always override derived definitions. See section 8.3 Derivation Rules for details.
  The `DerivedFrom` element has the following properties:
  - `typeRef`: The QName specifies the Relationship Type from which this Relationship Type derives its definitions.
- `PropertiesDefinition`: This element specifies the structure of the observable properties of the Relationship Type, such as its configuration and state, by means of XML schema.
  The `PropertiesDefinition` element has one but not both of the following properties:
  - `element`: This attribute provides the QName of an XML element defining the structure of the Relationship Type Properties.
  - `type`: This attribute provides the QName of an XML (complex) type defining the structure of the Relationship Type Properties.
- `InstanceStates`: This OPTIONAL element lists the set of states an instance of this Relationship Type can occupy at runtime. Those states are defined in nested `InstanceState` elements.
  The `InstanceState` element has the following nested properties:
  - `state`: This attribute specifies a URI that identifies a potential state.
- `SourceInterfaces`: This OPTIONAL element contains definitions of manageability interfaces that can be performed on the source of a relationship of this Relationship Type to actually establish the relationship between the source and the target in the deployed service.

 Copyright © OASIS Open 2013. All Rights Reserved.

Those interface definitions are contained in nested `Interface` elements, the content of which is that described for Node Type interfaces (see section 6.2).

- `TargetInterfaces`: This OPTIONAL element contains definitions of manageability interfaces that can be performed on the target of a relationship of this Relationship Type to actually establish the relationship between the source and the target in the deployed service.
  Those interface definitions are contained in nested `Interface` elements, the content of which is that described for Node Type interfaces (see section 6.2).

- `ValidSource`: This OPTIONAL element specifies the type of object that is allowed as a valid origin for relationships defined using the Relationship Type under definition. If not specified, any Node Type is allowed to be the origin of the relationship.
  The `ValidSource` element has the following properties:

  - `typeRef`: This attribute specifies the QName of a Node Type or Requirement Type that is allowed as a valid source for relationships defined using the Relationship Type under definition. Node Types or Requirements Types derived from the specified Node Type or Requirement Type, respectively, MUST also be accepted as valid relationship source.

    Note: If `ValidSource` specifies a Node Type, the `ValidTarget` element (if present) of the Relationship Type under definition MUST also specify a Node Type.
    If `ValidSource` specifies a Requirement Type, the `ValidTarget` element (if present) of the Relationship Type under definition MUST specify a Capability Type. This Capability Type MUST match the requirement defined in `ValidSource`, i.e. it MUST be of the type (or a sub-type of) the capability specified in the `requiredCapabilityType` attribute of the respective `RequirementType` definition.

- `ValidTarget`: This OPTIONAL element specifies the type of object that is allowed as a valid target for relationships defined using the Relationship Type under definition. If not specified, any Node Type is allowed to be the origin of the relationship.
  The `ValidTarget` element has the following properties:

  - `typeRef`: This attribute specifies the QName of a Node Type or Capability Type that is allowed as a valid target for relationships defined using the Relationship Type under definition. Node Types or Capability Types derived from the specified Node Type or Capability Type, respectively, MUST also be accepted as valid targets of relationships.

    Note: If `ValidTarget` specifies a Node Type, the `ValidSource` element (if present) of the Relationship Type under definition MUST also specify a Node Type.
    If `ValidTarget` specifies a Capability Type, the `ValidSource` element (if present) of the Relationship Type under definition MUST specify a Requirement Type. This Requirement Type MUST declare it requires the capability defined in `ValidTarget`, i.e. it MUST declare the type (or a super-type of) the capability in the `requiredCapabilityType` attribute of the respective `RequirementType` definition.

8.3 Derivation Rules

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

- Relationship Type Properties: It is assumed that the XML element (or type) representing the Relationship Type properties of the Relationship Type under definition extends the XML element (or type) of the Relationship Type properties of the Relationship Type referenced in the `DerivedFrom` element.

- Instance States: The resulting set of instance states of the Relationship Type under definition consists of the set union of the instances states defined by the Relationship Type derived from

 Copyright © OASIS Open 2013. All Rights Reserved.

and the instance states explicitly defined by the Relationship Type under definition. Instance states with the same state attribute will be combined into a single instance state of the same state.

- Valid source and target: An object specified as a valid source or target, respectively, of the Relationship Type under definition MUST be of a subtype defined as valid source or target, respectively, of the Relationship Type derived from.

  If the Relationship Type derived from has no valid source or target defined, the types of object being defined in the `ValidSource` or `ValidTarget` elements of the Relationship Type under definition are not restricted.

  If the Relationship Type under definition has no source or target defined, only the types of objects defined as source or target of the Relationship Type derived from are valid origins or destinations of the Relationship Type under definition.

- Interfaces: The set of interfaces (both source and target interfaces) of the Relationship Type under definition consists of the set union of interfaces defined by the Relationship Type derived from and the interfaces defined by the Relationship Type under definition.
  Two interfaces of the same name will be combined into a single, derived interface with the same name. The set of operations of the derived interface consists of the set union of operations defined by both interfaces. An operation defined by the Relationship Type under definition substitutes an operation with the same name of the Relationship Type derived from.

8.4 Example

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

```
01 <RelationshipType name="processDeployedOn">
02
03   <RelationshipTypeProperties element="ProcessDeployedOnProperties"/>
04
05   <InstanceStates>
06     <InstanceState state="www.example.com/successfullyDeployed"/>
07     <InstanceState state="www.example.com/failed"/>
08   </InstanceStates>
09
10 </RelationshipType>
```


Copyright © OASIS Open 2013. All Rights Reserved.

9 Relationship Type Implementations

This chapter specifies how *Relationship Type Implementations* are defined. A Relationship Type Implementation represents the runnable code that implements a specific Relationship Type. It provides a collection of executables implementing the interface operations of a Relationship Type (aka implementation artifacts). The particular executables are defined as separate Artifact Templates and are referenced from the implementation artifacts of a Relationship Type Implementation.

While Artifact Templates provide invariant information about an artifact – i.e. information that is context independent like the file name of the artifact – implementation artifacts can provide variant (or context specific) information, e.g. authentication data for a specific environment.

Relationship Type Implementations can specify hints for a TOSCA container that enable proper selection of an implementation that fits into a particular environment by means of Required Container Features definitions.

Note that there MAY be Relationship Types that do not define any interface operations, i.e. that also do not require any implementation artifacts. In such cases, no Relationship Type Implementation is needed but the respective Relationship Types can be used by a TOSCA implementation as is.

9.1 XML Syntax

The following pseudo schema defines the XML syntax of Relationship Type Implementations:

```
01 <RelationshipTypeImplementation name="xs:NCName"
02                                 targetNamespace="xs:anyURI"?
03                                 relationshipType="xs:QName"
04                                 abstract="yes|no"?
05                                 final="yes|no"?>
06
07   <Tags>
08     <Tag name="xs:string" value="xs:string"/> +
09   </Tags> ?
10
11   <DerivedFrom relationshipTypeImplementationRef="xs:QName"/> ?
12
13   <RequiredContainerFeatures>
14     <RequiredContainerFeature feature="xs:anyURI"/> +
15   </RequiredContainerFeatures> ?
16
17   <ImplementationArtifacts>
18     <ImplementationArtifact interfaceName="xs:NCName | xs:anyURI"?
19                             operationName="xs:NCName"?
20                             artifactType="xs:QName"
21                             artifactRef="xs:QName"?>
22         artifact specific content ?
23     <ImplementationArtifact> +
24   </ImplementationArtifacts> ?
25
26 </RelationshipTypeImplementation>
```

9.2 Properties

The `RelationshipTypeImplementation` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Relationship Type Implementation, which MUST be unique within the target namespace.

 Copyright © OASIS Open 2013. All Rights Reserved.

2080 • `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the
2081 definition of the Relationship Type Implementation will be added. If not specified, the Relationship
2082 Type Implementation will be added to the target namespace of the enclosing Definitions
2083 document.

2084 • `relationshipType`: The QName value of this attribute specifies the Relationship Type
2085 implemented by this Relationship Type Implementation.

2086 • `abstract`: This OPTIONAL attribute specifies that this Relationship Type Implementation
2087 cannot be used directly as an implementation for the Relationship Type specified in the
2088 `relationshipType` attribute.
2089
2090 For example, a Relationship Type implementer might decide to deliver only part of the
2091 implementation of a specific Relationship Type (i.e. for only some operations) for re-use purposes
2092 and require the implementation for speficic operations to be delivered in a more concrete, derived
2093 Relationship Type Implementation.
2094
2095 Note: an abstract Relationship Type Implementation MUST NOT be declared as final.

2096 • `final`: This OPTIONAL attribute specifies that other Relationship Type Implementations MUST
2097 NOT be derived from this Relationship Type Implementation.
2098
2099 Note: a final Relationship Type Implementation MUST NOT be declared as abstract.

2100 • `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by
2101 the author to describe the Relationship Type Implementation. Each tag is defined by a separate,
2102 nested `Tag` element.
2103 The `Tag` element has the following properties:

2104 o `name`: This attribute specifies the name of the tag.

2105 o `value`: This attribute specifies the value of the tag.
2106
2107 Note: The name/value pairs defined in tags have no normative interpretation.

2108 • `DerivedFrom`: This is an OPTIONAL reference to another Relationship Type Implementation
2109 from which this Relationship Type Implementation derives. See section 9.3 Derivation Rules or
2110 details.
2111 The `DerivedFrom` element has the following properties:

2112 o `relationshipTypeImplementationRef`: The QName specifies the Relationship
2113 Type Implementation from which this Relationship Type Implementation derives.

2114 • `RequiredContainerFeatures`: An implementation of a Relationship Type might depend
2115 on certain features of the environment it is executed in, such as specific (potentially proprietary)
2116 APIs of the TOSCA container.
2117 Thus, the contents of the `RequiredContainerFeatures` element provide “hints” to the
2118 TOSCA container allowing it to select the appropriate Relationship Type Implementation if
2119 multiple alternatives are provided.
2120 Each such dependency is defined by a separate `RequiredContainerFeature` element.
2121 The `RequiredContainerFeature` element has the following properties:

2122 o `feature`: The value of this attribute is a URI that denotes the corresponding needed
2123 feature of the environment.

2124 • `ImplementationArtifacts`: This element specifies a set of implementation artifacts for
2125 interfaces or operations of a Relationship Type.
2126 The `ImplementationArtifacts` element has the following properties:

2127 o `ImplementationArtifact`: This element specifies one implementation artifact of
2128 an interface or an operation.
2129

 Copyright © OASIS Open 2013. All Rights Reserved.

Note: Multiple implementation artifacts might be needed to implement a Relationship Type according to the attributes defined below. An implementation artifact MAY serve as implementation for all interfaces and all operations defined for the Relationship Type, it MAY serve as implementation for one interface (and all its operations), or it MAY serve as implementation for only one specific operation.

The `ImplementationArtifact` element has the following properties:

- `name`: This attribute specifies the name of the artifact, which SHOULD be unique within the scope of the encompassing Node Type Implementation.
- `interfaceName`: This OPTIONAL attribute specifies the name of the interface that is implemented by the actual implementation artifact. If not specified, the implementation artifact is assumed to provide the implementation for all interfaces defined by the Relationship Type referred to by the `relationshipType` attribute of the containing `RelationshipTypeImplementation`.

  Note that the referenced interface can be defined in either the `SourceInterfaces` element or the `TargetInterfaces` element of the Relationship Type implemented by this Relationship Type Implementation.
- `operationName`: This OPTIONAL attribute specifies the name of the operation that is implemented by the actual implementation artifact. If specified, the `interfaceName` MUST be specified and the specified `operationName` MUST refer to an operation of the specified interface. If not specified, the implementation artifact is assumed to provide the implementation for all operations defined within the specified interface.
- `artifactType`: This attribute specifies the type of this artifact. The QName value of this attribute SHOULD correspond to the QName of an `ArtifactType` defined in the same Definitions document or in an imported document.

  The `artifactType` attribute specifies the artifact type specific content of the `ImplementationArtifact` element body and indicates the type of Artifact Template referenced by the Implementation Artifact via the `artifactRef` attribute.
- `artifactRef`: This OPTIONAL attribute contains a QName that identifies an Artifact Template to be used as implementation artifact. This Artifact Template can be defined in the same Definitions document or in a separate, imported document.
  The type of Artifact Template referenced by the `artifactRef` attribute MUST be the same type or a sub-type of the type specified in the `artifactType` attribute.

  Note: if no Artifact Template is referenced, the artifact type specific content of the `ImplementationArtifact` element alone is assumed to represent the actual artifact. For example, a simple script could be defined in place within the `ImplementationArtifact` element.

9.3 Derivation Rules

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

- Implementation Artifacts: The set of implementation artifacts of a Relationship Type Implementation consists of the set union of implementation artifacts defined by the Relationship

 Copyright © OASIS Open 2013. All Rights Reserved.

Type Implementation itself and the implementation artifacts defined by any Relationship Type Implemenation the Relationship Type Implementation is derived from.

An implementation artifact defined by a Node Type Implementation overrides an implementation artifact having the same interface name and operation name of a Relationship Type Implementation the Relationship Type Implementation is derived from.

If an implementation artifact defined in a Relationship Type Implementation specifies only an interface name, it substitutes implementation artifacts having the same interface name (with or without an operation name defined) of any Relationship Type Implementation the Relationship Type Implementation is derived from. In this case, the implementation of a complete interface of a Relationship Type is overridden.

If an implementation artifact defined in a Relationship Type Implementation neither defines an interface name nor an operation name, it overrides all implementation artifacts of any Relationship Type Implementation the Relationship Type Implementation is derived from. In this case, the complete implementation of a Relationship Type is overridden.

9.4 Example

The following example defines the Node Type Implementation “MyDBMSImplementation”. This is an implementation of a Node Type “DBMS”.

```
01 <Definitions id="MyImpls" name="My Implementations"
02   targetNamespace="http://www.example.com/SampleImplementations"
03   xmlns:bn="http://www.example.com/BaseRelationshipTypes"
04   xmlns:ba="http://www.example.com/BaseArtifactTypes"
05   xmlns:sa="http://www.example.com/SampleArtifacts">
06
07   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
08           namespace="http://www.example.com/BaseArtifactTypes"/>
09
10   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
11           namespace="http://www.example.com/BaseRelationshipTypes"/>
12
13   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
14           namespace="http://www.example.com/SampleArtifacts"/>
15
16   <RelationshipTypeImplementation name="MyDBConnectImplementation"
17                                   relationshipType="bn:DBConnection">
18
19      <ImplementationArtifacts>
20        <ImplementationArtifact interfaceName="ConnectionInterface"
21                                operationName="connectTo"
22                                artifactType="ba:ScriptArtifact"
23                                artifactRef="sa:MyConnectScript">
24        <ImplementationArtifact>
25      </ImplementationArtifacts>
26
27   </RelationshipTypeImplementation>
28
29 </Definitions>
```

The Relationship Type Implementation contains the “MyDBConnectionImpl” implementation artifact, which is an artifact for the “ConnectionInterface” interface that has been defined for the “DBConnection” base Relationship Type. The type of this artifact is a “ScriptArtifact” that has been defined as base Artifact Type. The implementation artifact refers to the “MyConnectScript” Artifact Template that has been defined before.

Copyright © OASIS Open 2013. All Rights Reserved.

10 Requirement Types

This chapter specifies how *Requirement Types* are defined. A Requirement Type is a reusable entity that describes a kind of requirement that a Node Type can declare to expose. For example, a Requirement Type for a database connection can be defined and various Node Types (e.g. a Node Type for an application) can declare to expose (or “to have”) a requirement for a database connection.

A Requirement Type defines the structure of observable properties via a *Properties Definition*, i.e. the names, data types and allowed values the properties defined in *Requirements* of Node Templates of a Node Type can have in cases where the Node Type defines a requirement of the respective Requirement Type.

A Requirement Type can inherit properties and semantics from another Requirement Type by means of the `DerivedFrom` element. Requirement Types might be declared as abstract, meaning that they cannot be instantiated. The purpose of such abstract Requirement Types is to provide common properties for re-use in specialized, derived Requirement Types. Requirement Types might also be declared as final, meaning that they cannot be derived by other Requirement Types.

10.1 XML Syntax

The following pseudo schema defines the XML syntax of Requirement Types:

```
01 <RequirementType name="xs:NCName"
02                  targetNamespace="xs:anyURI"?
03                  abstract="yes|no"?
04                  final="yes|no"?
05                  requiredCapabilityType="xs:QName"?>
06
07   <Tags>
08     <Tag name="xs:string" value="xs:string"/> +
09   </Tags> ?
10
11   <DerivedFrom typeRef="xs:QName"/> ?
12
13   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
14
15 </RequirementType>
```

10.2 Properties

The `RequirementType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Requirement Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Requirement Type will be added. If not specified, the Requirement Type definition will be added to the target namespace of the enclosing Definitions document.
- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Node Templates of a Node Type that defines a requirement of this Requirement Type.

  As a consequence, a Node Type with a Requirement Definition of an abstract Requirement Type MUST be declared as abstract as well and a derived Node Type that defines a requirement of a type derived from the abstract Requirement Type has to be defined. For example, an abstract Node Type “Application” might be defined having a requirement of the abstract type “Container”. A derived Node Type “Web Application” can then be defined with a more concrete requirement of type “Web Application Container” which can then be used for defining Node Templates that can

 Copyright © OASIS Open 2013. All Rights Reserved.

be instantiated during the creation of a service according to a Service Template.

Note: an abstract Requirement Type MUST NOT be declared as final.

- `final`: This OPTIONAL attribute specifies that other Requirement Types MUST NOT be derived from this Requirement Type.

  Note: a final Requirement Type MUST NOT be declared as abstract.

- `requiredCapabilityType`; This OPTIONAL attribute specifies the type of capability needed to match the defined Requirement Type. The QName value of this attribute refers to the QName of a `CapabilityType` element defined in the same Definitions document or in a separate, imported document.

  Note: The following basic match-making for Requirements and Capabilities MUST be supported by each TOSCA implementation. Each Requirement is defined by a Requirement Definition, which in turn refers to a Requirement Type that specifies the needed Capability Type by means of its `requiredCapabilityType` attribute. The value of this attribute is used for basic type-based match-making: a Capability matches a Requirement if the Requirement's Requirement Type has a `requiredCapabilityType` value that corresponds to the Capability Type of the Capability or one of its super-types.
  Any domain-specific match-making semantics (e.g. based on constraints or properties) has to be defined in the cause of specifying the corresponding Requirement Types and Capability Types.

- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Requirement Type. Each tag is defined by a separate, nested `Tag` element.
  The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.

- `DerivedFrom`: This is an OPTIONAL reference to another Requirement Type from which this Requirement Type derives. See section 10.3 Derivation Rules for details.
  The `DerivedFrom` element has the following properties:
  - `typeRef`: The QName specifies the Requirement Type from which this Requirement Type derives its definitions and semantics.
- `PropertiesDefinition`: This element specifies the structure of the observable properties of the Requirement Type, such as its configuration and state, by means of XML schema.
  The `PropertiesDefinition` element has one but not both of the following properties:
  - `element`: This attribute provides the QName of an XML element defining the structure of the Requirement Type Properties.
  - `type`: This attribute provides the QName of an XML (complex) type defining the structure of the Requirement Type Properties.

10.3 Derivation Rules

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

- Requirement Type Properties: It is assumed that the XML element (or type) representing the Requirement Type Properties extends the XML element (or type) of the Requirement Type Properties of the Requirement Type referenced in the `DerivedFrom` element.

 Copyright © OASIS Open 2013. All Rights Reserved.

10.4 Example

The following example defines the Requirement Type “DatabaseClientEndpoint” that expresses the requirement of a client for a database connection. It is defined in a Definitions document “MyRequirements” within the target namespace “http://www.example.com/SampleRequirements”. Thus, by importing the corresponding namespace into another Definitions document, the “DatabaseClientEndpoint” Requirement Type is available for use in the other document.

```
01 <Definitions id="MyRequirements" name="My Requirements"
02   targetNamespace="http://www.example.com/SampleRequirements"
03   xmlns:br="http://www.example.com/BaseRequirementTypes"
04   xmlns:mrp="http://www.example.com/SampleRequirementProperties>
05
06   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
07     namespace="http://www.example.com/BaseRequirementTypes"/>
08
09   <Import importType="http://www.w3.org/2001/XMLSchema"
10     namespace="http://www.example.com/SampleRequirementProperties"/>
11
12   <RequirementType name="DatabaseClientEndpoint">
13     <DerivedFrom typeRef="br:ClientEndpoint"/>
14     <PropertiesDefinition
15       element="mrp:DatabaseClientEndpointProperties"/>
16   </RequirementType>
17
18 </Definitions>
```

The Requirement Type “DatabaseClientEndpoint” defined in the example above is derived from another generic “ClientEndpoint” Requirement Type defined in a separate file by means of the `DerivedFrom` element. The definitions in that separate Definitions file are imported by means of the first `Import` element and the namespace of those imported definitions is assigned the prefix `“br”` in the current file.

The “DatabaseClientEndpoint” Requirement Type defines a set of properties through an XML schema element definition “DatabaseClientEndpointProperties”. For example, those properties might include the definition of a port number to be used for client connections. The XML schema definition is stored in a separate XSD file that is imported by means of the second `Import` element. The namespace of the XML schema definitions is assigned the prefix `“mrp”` in the current file.

 Copyright © OASIS Open 2013. All Rights Reserved.

11 Capability Types

This chapter specifies how *Capability Types* are defined. A Capability Type is a reusable entity that describes a kind of capability that a Node Type can declare to expose. For example, a Capability Type for a database server endpoint can be defined and various Node Types (e.g. a Node Type for a database) can declare to expose (or to "provide") the capability of serving as a database server endpoint.

A Capability Type defines the structure of observable properties via a *Properties Definition*, i.e. the names, data types and allowed values the properties defined in *Capabilities* of Node Templates of a Node Type can have in cases where the Node Type defines a capability of the respective Capability Type.

A Capability Type can inherit properties and semantics from another Capability Type by means of the `DerivedFrom` element. Capability Types might be declared as abstract, meaning that they cannot be instantiated. The purpose of such abstract Capability Types is to provide common properties for re-use in specialized, derived Capability Types. Capability Types might also be declared as final, meaning that they cannot be derived by other Capability Types.

11.1 XML Syntax

The following pseudo schema defines the XML syntax of Capability Types:

```
01 <CapabilityType name="xs:NCName"
02                 targetNamespace="xs:anyURI"?
03                 abstract="yes|no"?
04                 final="yes|no"?>
05
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <DerivedFrom typeRef="xs:QName"/> ?
11
12   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
13
14 </CapabilityType>
```

11.2 Properties

The `CapabilityType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Capability Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Capability Type will be added. If not specified, the Capability Type definition will be added to the target namespace of the enclosing Definitions document.
- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Node Templates of a Node Type that defines a capability of this Capability Type.

  As a consequence, a Node Type with a Capability Definition of an abstract Capability Type MUST be declared as abstract as well and a derived Node Type that defines a capability of a type derived from the abstract Capability Type has to be defined. For example, an abstract Node Type "Server" might be defined having a capability of the abstract type "Container". A derived Node Type "Web Server" can then be defined with a more concrete capability of type "Web Application Container" which can then be used for defining Node Templates that can be instantiated during the creation of a service according to a Service Template.

Copyright © OASIS Open 2013. All Rights Reserved.

Note: an abstract Capability Type MUST NOT be declared as final.

- `final`: This OPTIONAL attribute specifies that other Capability Types MUST NOT be derived from this Capability Type.

  Note: a final Capability Type MUST NOT be declared as abstract.

- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Capability Type. Each tag is defined by a separate, nested `Tag` element.
  The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.

- `DerivedFrom`: This is an OPTIONAL reference to another Capability Type from which this Capability Type derives. See section 11.3 Derivation Rules for details.
  The `DerivedFrom` element has the following properties:
  - `typeRef`: The QName specifies the Capability Type from which this Capability Type derives its definitions and semantics.

- `PropertiesDefinition`: This element specifies the structure of the observable properties of the Capability Type, such as its configuration and state, by means of XML schema.
  The `PropertiesDefinition` element has one but not both of the following properties:
  - `element`: This attribute provides the QName of an XML element defining the structure of the Capability Type Properties.
  - `type`: This attribute provides the QName of an XML (complex) type defining the structure of the Capability Type Properties.

11.3 Derivation Rules

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

- Capability Type Properties: It is assumed that the XML element (or type) representing the Capability Type Properties extends the XML element (or type) of the Capability Type Properties of the Capability Type referenced in the `DerivedFrom` element.

11.4 Example

The following example defines the Capability Type “DatabaseServerEndpoint” that expresses the capability of a component to serve database connections. It is defined in a Definitions document “MyCapabilities” within the target namespace “http://www.example.com/SampleCapabilities”. Thus, by importing the corresponding namespace into another Definitions document, the “DatabaseServerEndpoint” Capability Type is available for use in the other document.

```
01 <Definitions id="MyCapabilities" name="My Capabilities"
02   targetNamespace="http://www.example.com/SampleCapabilities"
03   xmlns:bc="http://www.example.com/BaseCapabilityTypes"
04   xmlns:mcp="http://www.example.com/SampleCapabilityProperties>
05
06   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
07           namespace="http://www.example.com/BaseCapabilityTypes"/>
08
09   <Import importType="http://www.w3.org/2001/XMLSchema"
10           namespace="http://www.example.com/SampleCapabilityProperties"/>
```


Copyright © OASIS Open 2013. All Rights Reserved.

2450
2451
2452
2453
2454
2455
2456
2457

```
11
12   <CapabilityType name="DatabaseServerEndpoint">
13     <DerivedFrom typeRef="bc:ServerEndpoint"/>
14     <PropertiesDefinition
15       element="mcp:DatabaseServerEndpointProperties"/>
16   </CapabilityType>
17
18 </Definitions>
```

2458 The Capability Type “DatabaseServerEndpoint” defined in the example above is derived from another
2459 generic “ServerEndpoint” Capability Type defined in a separate file by means of the `DerivedFrom`
2460 element. The definitions in that separate Definitions file are imported by means of the first `Import`
2461 element and the namespace of those imported definitions is assigned the prefix “`bc`” in the current file.

2462 The “DatabaseServerEndpoint” Capability Type defines a set of properties through an XML schema
2463 element definition “DatabaseServerEndpointProperties”. For example, those properties might include the
2464 definition of a port number where the server listens for client connections, or credentials to be used by
2465 clients. The XML schema definition is stored in a separate XSD file that is imported by means of the
2466 second `Import` element. The namespace of the XML schema definitions is assigned the prefix “`mcp`”
2467 in the current file.

 Copyright © OASIS Open 2013. All Rights Reserved.

12 Artifact Types

This chapter specifies how *Artifact Types* are defined. An Artifact Type is a reusable entity that defines the type of one or more Artifact Templates which in turn serve as deployment artifacts for Node Templates or implementation artifacts for Node Type and Relationship Type interface operations. For example, an Artifact Type “WAR File” might be defined for describing web application archive files. Based on this Artifact Type, one or more Artifact Templates representing concrete WAR files can be defined and referenced as deployment or implementation artifacts.

An Artifact Type can define the structure of observable properties via a *Properties Definition*, i.e. the names, data types and allowed values the properties defined in Artifact Templates using an Artifact Type or instances of such Artifact Templates can have. Note that properties defined by an Artifact Type are assummed to be invariant across the contexts in which corresponding artifacts are used – as opposed to properties that can vary depending on the context. As an example of such an invariant property, an Artifact Type for a WAR file could define a “signature” property that can hold a hash for validating the actual artifact proper. In contrast, the path where the web application contained in the WAR file gets deployed can vary for each place where the WAR file is used.

An Artifact Type can inherit definitions and semantics from another Artifact Type by means of the `DerivedFrom` element. Artifact Types can be declared as abstract, meaning that they cannot be instantiated. The purpose of such abstract Artifact Types is to provide common properties for re-use in specialized, derived Artifact Types. Artifact Types can also be declared as final, meaning that they cannot be derived by other Artifact Types.

12.1 XML Syntax

The following pseudo schema defines the XML syntax of Artifact Types:

```
01 <ArtifactType name="xs:NCName"
02               targetNamespace="xs:anyURI"?
03               abstract="yes|no"?
04               final="yes|no"?>
05
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <DerivedFrom typeRef="xs:QName"/> ?
11
12   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
13
14 </ArtifactType>
```

12.2 Properties

The `ArtifactType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Artifact Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Artifact Type will be added. If not specified, the Artifact Type definition will be added to the target namespace of the enclosing Definitions document.
- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Artifact Templates of that abstract Artifact Type, i.e. the respective artifacts cannot be used directly as deployment or implementation artifact in any context.

 Copyright © OASIS Open 2013. All Rights Reserved.

As a consequence, an Artifact Template of an abstract Artifact Type MUST be replaced by an artifact of a derived Artifact Type at the latest during deployment of the element that uses the artifact (i.e. a Node Template or Relationship Template).

Note: an abstract Artifact Type MUST NOT be declared as final.

- `final`: This OPTIONAL attribute specifies that other Artifact Types MUST NOT be derived from this Artifact Type.

  Note: a final Artifact Type MUST NOT be declared as abstract.
- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Artifact Type. Each tag is defined by a separate, nested `Tag` element. The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.
- `DerivedFrom`: This is an OPTIONAL reference to another Artifact Type from which this Artifact Type derives. See section 12.3 Derivation Rules for details.
  The `DerivedFrom` element has the following properties:
  - `typeRef`: The QName specifies the Artifact Type from which this Artifact Type derives its definitions and semantics.
- `PropertiesDefinition`: This element specifies the structure of the observable properties of the Artifact Type, such as its configuration and state, by means of XML schema.
  The `PropertiesDefinition` element has one but not both of the following properties:
  - `element`: This attribute provides the QName of an XML element defining the structure of the Artifact Type Properties.
  - `type`: This attribute provides the QName of an XML (complex) type defining the structure of the Artifact Type Properties.

12.3 Derivation Rules

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

- Artifact Type Properties: It is assumed that the XML element (or type) representing the Artifact Type Properties extends the XML element (or type) of the Artifact Type Properties of the Artifact Type referenced in the `DerivedFrom` element.

12.4 Example

The following example defines the Artifact Type “RPMPackage” that can be used for describing RPM packages as deployable artifacts on various Linux distributions. It is defined in a Definitions document “MyArtifacts” within the target namespace “http://www.example.com/SampleArtifacts”. Thus, by importing the corresponding namespace into another Definitions document, the “RPMPackage” Artifact Type is available for use in the other document.

```
01 <Definitions id="MyArtifacts" name="My Artifacts"
02   targetNamespace="http://www.example.com/SampleArtifacts"
03   xmlns:ba="http://www.example.com/BaseArtifactTypes"
04   xmlns:map="http://www.example.com/SampleArtifactProperties>
05
06   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
07           namespace="http://www.example.com/BaseArtifactTypes"/>
08
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
09   <Import importType="http://www.w3.org/2001/XMLSchema"
10           namespace="http://www.example.com/SampleArtifactProperties"/>
11
12   <ArtifactType name="RPMPackage">
13     <DerivedFrom typeRef="ba:OSPackage"/>
14     <PropertiesDefinition element="map:RPMPackageProperties"/>
15   </ArtifactType>
16
17 </Definitions>
```

The Artifact Type “RPMPackage” defined in the example above is derived from another generic “OSPackage” Artifact Type defined in a separate file by means of the `DerivedFrom` element. The definitions in that separate Definitions file are imported by means of the first `Import` element and the namespace of those imported definitions is assigned the prefix “`ba`” in the current file.

The “RPMPackage” Artifact Type defines a set of properties through an XML schema element definition “RPMPackageProperties”. For example, those properties might include the definition of the name or names of one or more RPM packages. The XML schema definition is stored in a separate XSD file that is imported by means of the second `Import` element. The namespace of the XML schema definitions is assigned the prefix “`map`” in the current file.

 Copyright © OASIS Open 2013. All Rights Reserved.

13 Artifact Templates

This chapter specifies how *Artifact Templates* are defined. An Artifact Template represents an artifact that can be referenced from other objects in a Service Template as a deployment artifact or implementation artifact. For example, from Node Types or Node Templates, an Artifact Template for some software installable could be referenced as a deployment artifact for materializing a specific software component. As another example, from within interface definitions of Node Types or Relationship Types, an Artifact Template for a WAR file could be referenced as implementation artifact for a REST operation.

An Artifact Template refers to a specific Artifact Type that defines the structure of observable properties (metadata) or the artifact. The Artifact Template then typically defines values for those properties inside the `Properties` element. Note that properties defined by an Artifact Type are asumed to be invariant across the contexts in which corresponding artifacts are used – as opposed to properties that can vary depending on the context.

Furthermore, an Artifact Template typically provides one or more references to the actual artifact itself that can be contained as a file in the CSAR (see section 3.7 and section 14) containing the overall Service Template or that can be available at a remote location such as an FTP server.

13.1 XML Syntax

The following pseudo schema defines the XML syntax of Artifact Templates:

```
01 <ArtifactTemplate id="xs:ID" name="xs:string"? type="xs:QName">
02
03   <Properties>
04     XML fragment
05   </Properties> ?
06
07   <PropertyConstraints>
08     <PropertyConstraint property="xs:string"
09                         constraintType="xs:anyURI"> +
10       constraint ?
11     </PropertyConstraint>
12   </PropertyConstraints> ?
13
14   <ArifactReferences>
15     <ArtifactReference reference="xs:anyURI">
16       (
17        <Include pattern="xs:string"/>
18       |
19        <Exclude pattern="xs:string"/>
20       )*
21     </ArtifactReference> +
22   </ArtifactReferences> ?
23
24 </ArtifactTemplate>
```

13.2 Properties

The `ArtifactTemplate` element has the following properties:

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

 Copyright © OASIS Open 2013. All Rights Reserved.

- `type`: The QName value of this attribute refers to the Artifact Type providing the type of the Artifact Template.

  Note: If the Artifact Type referenced by the `type` attribute of an Artifact Template is declared as abstract, no instances of the specific Artifact Template can be created, i.e. the artifact cannot be used directly as deployment or implementation artifact. Instead, a substitution of the Artifact Template with one having a specialized, derived Artifact Type has to be done at the latest during the instantiation time of a Service Template.

- `Properties`: This OPTIONAL element specifies the invariant properties of the Artifact Template, i.e. those properties that will be commonly used across different contexts in which the Artifact Template is used.

  The initial values are specified by providing an instance document of the XML schema of the corresponding Artifact Type Properties. This instance document considers the inheritance structure deduced by the `DerivedFrom` property of the Artifact Type referenced by the type attribute of the Artifact Template.

- `PropertyConstraints`: This OPTIONAL element specifies constraints on the use of one or more of the Artifact Type Properties of the Artifact Type providing the property definitions for the Artifact Template. Each constraint is specified by means of a separate nested `PropertyConstraint` element.
  The `PropertyConstraint` element has the following properties:

  - `property`: The string value of this property is an XPath expression pointing to the property within the Artifact Type Properties document that is constrained within the context of the Artifact Template. More than one constraint MUST NOT be defined for each property.

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

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

- `ArtifactReferences`: This OPTIONAL element contains one or more references to the actual artifact proper, each represented by a separate `ArtifactReference` element.
  The `ArtifactReference` element has the following properties:

  - `reference`: This attribute contains a URI pointing to an actual artifact. If this URI is a relative URI, it is interpreted relative to the root directory of the CSAR containing the Service Template (see also sections 3.7 and 14).

  - `Include`: This OPTIONAL element can be used to define a pattern of files that are to be included in the artifact reference in case the reference points to a complete directory.
    The `Include` element has the following properties:

    - `pattern`: This attribute contains a pattern definition for files that are to be included in the overall artifact reference. For example, a pattern of `"*.py"` would include all python scripts contained in a directory.

  - `Exclude`: This OPTIONAL element can be used to define a pattern of files that are to be excluded from the artifact reference in case the reference points to a complete directory.
    The `Exclude` element has the following properties:

 Copyright © OASIS Open 2013. All Rights Reserved.

2675 ▪ `pattern`: This attribute contains a pattern definition for files that are to be
2676 excluded in the overall artifact reference. For example, a pattern of "`*.sh`"
2677 would exclude all bash scripts contained in a directory.

2678 13.3 Example

2679 The following example defines the Artifact Template “MyInstallable” that points to a zip file containing
2680 some software installable. It is defined in a Definitions document “MyArtifacts” within the target
2681 namespace “http://www.example.com/SampleArtifacts”. The Artifact Template can be used in the same
2682 document, for example as a deployment artifact for some Node Template representing a software
2683 component, or it can be used in other Definitions documents by importing the corresponding namespace
2684 into another document.

```
2685 01 <Definitions id="MyArtifacts" name="My Artifacts"
2686 02   targetNamespace="http://www.example.com/SampleArtifacts"
2687 03   xmlns:ba="http://www.example.com/BaseArtifactTypes">
2688 04
2689 05   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
2690 06           namespace="http://www.example.com/BaseArtifactTypes"/>
2691 07
2692 08   <ArtifactTemplate id="MyInstallable"
2693 09                     name="My installable"
2694 10                     type="ba:ZipFile">
2695 11     <ArtifactReferences>
2696 12       <ArtifactReference reference="files/MyInstallable.zip"/>
2697 13     </ArtifactReferences>
2698 14   </ArtifactTemplate>
2699 15
2700 16 </Definitions>
```

2701 The Artifact Template “MyInstallable” defined in the example above is of type “ZipFile” that is specified in
2702 the `type` attribute of the `ArtifactTemplate` element. This Artifact Type is defined in a separate file,
2703 the definitions of which are imported by means of the `Import` element and the namespace of those
2704 imported definitions is assigned the prefix "`ba`" in the current file.

2705 The “MyInstallable” Artifact Template provides a reference to a file “MyInstallable.zip” by means of the
2706 `ArtifactReference` element. Since the URI provided in the `reference` attribute is a relative URI,
2707 it is interpreted relative to the root directory of the CSAR containing the Service Template.

 Copyright © OASIS Open 2013. All Rights Reserved.

14 Policy Types

This chapter specifies how *Policy Types* are defined. A Policy Type is a reusable entity that describes a kind of non-functional behavior or a kind of quality-of-service (QoS) that a Node Type can declare to expose. For example, a Policy Type can be defined to express high availability for specific Node Types (e.g. a Node Type for an application server).

A Policy Type defines the structure of observable properties via a Properties Definition, i.e. the names, data types and allowed values the properties defined in a corresponding Policy Template can have.

A Policy Type can inherit properties from another Policy Type by means of the `DerivedFrom` element.

A Policy Type declares the set of Node Types it specifies non-functional behavior for via the `AppliesTo` element. Note that being "applicable to" does not enforce implementation: i.e. in case a Policy Type expressing high availability is associated with a "Webserver" Node Type, an instance of the Webserver is not necessarily highly available. Whether or not an instance of a Node Type to which a Policy Type is applicable will show the specified non-functional behavior, is determined by a Node Template of the corresponding Node Type.

14.1 XML Syntax

The following pseudo schema defines the XML syntax of Policy Types:

```
01 <PolicyType name="xs:NCName"
02             policyLanguage="xs:anyURI"?
03             abstract="yes|no"?
04             final="yes|no"?
05             targetNamespace="xs:anyURI"?>
06   <Tags>
07     <Tag name="xs:string" value="xs:string"/> +
08   </Tags> ?
09
10   <DerivedFrom typeRef="xs:QName"/> ?
11
12   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
13
14   <AppliesTo>
15     <NodeTypeReference typeRef="xs:QName"/> +
16   </AppliesTo> ?
17
18   policy type specific content ?
19
20 </PolicyType>
```

14.2 Properties

The `PolicyType` element has the following properties:

- `name`: This attribute specifies the name or identifier of the Policy Type, which MUST be unique within the target namespace.
- `targetNamespace`: This OPTIONAL attribute specifies the target namespace to which the definition of the Policy Type will be added. If not specified, the Policy Type definition will be added to the target namespace of the enclosing Definitions document.
- `policyLanguage`: This OPTIONAL attribute specifies the language used to specify the details of the Policy Type. These details can be defined as policy type specific content of the PolicyType element.

 Copyright © OASIS Open 2013. All Rights Reserved.

- `abstract`: This OPTIONAL attribute specifies that no instances can be created from Policy Templates of that abstract Policy Type, i.e. the respective policies cannot be used directly during the instantiation of a Service Template.

  As a consequence, a Policy Template of an abstract Policy Type MUST be replaced by a policy of a derived Policy Type at the latest during deployment of the element that policy is attached to.
- `final`: This OPTIONAL attribute specifies that other Policy Types MUST NOT be derived from this Policy Type.

  Note: a final Policy Type MUST NOT be declared as abstract.
- `Tags`: This OPTIONAL element allows the definition of any number of tags which can be used by the author to describe the Policy Type. Each tag is defined by a separate, nested `Tag` element. The `Tag` element has the following properties:
  - `name`: This attribute specifies the name of the tag.
  - `value`: This attribute specifies the value of the tag.

    Note: The name/value pairs defined in tags have no normative interpretation.
- `DerivedFrom`: This is an OPTIONAL reference to another Policy Type from which this Policy Type derives. See section 14.3 Derivation Rules for details.
  The `DerivedFrom` element has the following properties:
  - `typeRef`: The QName specifies the Policy Type from which this Policy Type derives its definitions from.
- `PropertiesDefinition`: This element specifies the structure of the observable properties of the Policy Type by means of XML schema.
  The `PropertiesDefinition` element has one but not both of the following properties:
  - `element`: This attribute provides the QName of an XML element defining the structure of the Policy Type Properties.
  - `type`: This attribute provides the QName of an XML (complex) type defining the structure of the Policy Type Properties.
- `AppliesTo`: This OPTIONAL element specifies the set of Node Types the Policy Type is applicable to, each defined as a separate, nested `NodeTypeReference` element.
  The `NodeTypeReference` element has the following property:
  - `typeRef`: The attribute provides the QName of a Node Type to which the Policy Type applies.

14.3 Derivation Rules

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

- Properties Definitions: It is assumed that the XML element (or type) representing the Policy Type Properties Definitions extends the XML element (or type) of the Policy Type Properties Definitions of the Policy Type referenced in the `DerivedFrom` element.
- Applies To: The set of Node Types the Policy Type is applicable to consist of the set union of Node Types derived from and Node Types explicitly referenced by the Policy Type by means of its `AppliesTo` element.
- Policy Language: A Policy Type MUST define the same policy language as the Policy Type it derives from. In case the Policy Type used as basis for derivation has no `policyLanguage` attribute defined, the deriving Policy Type can define any appropriate policy language.

 Copyright © OASIS Open 2013. All Rights Reserved.

14.4 Example

The following example defines two Policy Types, the “HighAvailability” Policy Type and the “ContinuousAvailability” Policy Type. They are defined in a Definitions document “MyPolicyTypes” within the target namespace “http://www.example.com/SamplePolicyTypes”. Thus, by importing the corresponding namespace into another Definitions document, both Policy Types are available for use in the other document.

```
01 <Definitions id="MyPolicyTypes" name="My Policy Types"
02   targetNamespace="http://www.example.com/SamplePolicyTypes"
03   xmlns:bnt="http://www.example.com/BaseNodeTypes">
04   xmlns:spp="http://www.example.com/SamplePolicyProperties">
05
06   <Import importType="http://www.w3.org/2001/XMLSchema"
07           namespace="http://www.example.com/SamplePolicyProperties"/>
08
09   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
10           namespace="http://www.example.com/BaseNodeTypes"/>
11
12
13   <PolicyType name="HighAvailability">
14     <PropertiesDefinition element="spp:HAProperties"/>
15   </PolicyType>
16
17   <PolicyType name="ContinuousAvailability">
18     <DerivedFrom typeRef="HighAvailability"/>
19     <PropertiesDefinition element="spp:CAProperties"/>
20     <AppliesTo>
21       <NodeTypeReference typeRef="bnt:DBMS"/>
22     </AppliesTo>
23   </PolicyType>
24
25 </Definitions>
```

The Policy Type “HighAvailability” defined in the example above has the “HAProperties” properties that are defined in a separate namespace as an XML element. The same namespace contains the “CAProperties” element that defines the properties of the “ContinuousAvailability” Policy Type. This namespace is imported by means of the first `Import` element and the namespace of those imported definitions is assigned the prefix “spp” in the current file.

The “ContinuousAvailability” Policy Type is derived from the “HighAvailability” Policy Type. Furthermore, it is applicable to the “DBMS” Node Type. This Node Type is defined in a separate namespace, which is imported by means of the second `Import` element and the namespace of those imported definitions is assigned the prefix “bnt” in the current file.

Copyright © OASIS Open 2013. All Rights Reserved.

15 Policy Templates

This chapter specifies how *Policy Templates* are defined. A Policy Template represents a particular non-functional behavior or quality-of-service that can be referenced by a Node Template. A Policy Template refers to a specific Policy Type that defines the structure of observable properties (metadata) of the non-functional behavior. The Policy Template then typically defines values for those properties inside the `Properties` element. Note that properties defined by a Policy Template are assumed to be invariant across the contexts in which corresponding behavior is exposed – as opposed to properties defined in Policies of Node Templates that may vary depending on the context.

15.1 XML Syntax

The following pseudo schema defines the XML syntax of Policy Templates:

```
01 <PolicyTemplate id="xs:ID" name="xs:string"? type="xs:QName">
02
03   <Properties>
04     XML fragment
05   </Properties> ?
06
07   <PropertyConstraints>
08     <PropertyConstraint property="xs:string"
09                         constraintType="xs:anyURI"> +
10       constraint ?
11     </PropertyConstraint>
12   </PropertyConstraints> ?
13
14   policy type specific content ?
15
16 </PolicyTemplate>
```

15.2 Properties

The `PolicyTemplate` element has the following properties:

- `id`: This attribute specifies the identifier of the Policy Template which MUST be unique within the target namespace.
- `name`: This OPTIONAL attribute specifies the name of the Policy Template.
- `type`: The QName value of this attribute refers to the Policy Type providing the type of the Policy Template.
- `Properties`: This OPTIONAL element specifies the invariant properties of the Policy Template, i.e. those properties that will be commonly used across different contexts in which the Policy Template is used.

  The initial values are specified by providing an instance document of the XML schema of the corresponding Policy Type Properties. This instance document considers the inheritance structure deduced by the DerivedFrom property of the Policy Type referenced by the type attribute of the Policy Template.
- `PropertyConstraints`: This OPTIONAL element specifies constraints on the use of one or more of the Policy Type Properties of the Policy Type providing the property definitions for the Policy Template. Each constraint is specified by means of a separate nested `PropertyConstraint` element.
  The `PropertyConstraint` element has the following properties:

Copyright © OASIS Open 2013. All Rights Reserved.

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

15.3 Example

The following example defines a Policy Template “MyHAPolicy”. It is defined in a Definitions document “MyPolicies” within the target namespace “http://www.example.com/SamplePolicies”. The Policy Template can be used in the same Definitions document, for example, as a Policy of some Node Template, or it can be used in other document by importing the corresponding namespace into the other document.

```
01 <Definitions id="MyPolices" name="My Policies"
02   targetNamespace="http://www.example.com/SamplePolicies"
03   xmlns:spt="http://www.example.com/SamplePolicyTypes">
04
05   <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
06           namespace="http://www.example.com/SamplePolicyTypes"/>
07
08   <PolicyTemplate id="MyHAPolicy"
09                   name="My High Availability Policy"
10                   type="bpt:HighAvailability">
11     <Properties>
12       <HAProperties>
13         <AvailabilityClass>4</AvailabilityClass>
14         <HeartbeatFrequency measuredIn="msec">
15            250
16         </HeartbeatFrequency>
17       </HAProperties>
18     </Properties>
19   </PolicyTemplate>
20
21 </Definitions>
```

The Policy Template “MyHAPolicy” defined in the example above is of type “HighAvailability” that is specified in the `type` attribute of the `PolicyTemplate` element. This Policy Type is defined in a separate file, the definitions of which are imported by means of the `Import` element and the namespace of those imported definitions is assigned the prefix “spt” in the current file.

The “MyHAPolicy” Policy Template provides values for the properties defined by the Properties Definition of the “HighAvailability” Policy Type. The AvailabilityClass property is set to “4”. The value of the HeartbeatFrequency is “250”, measured in “msec”.

Copyright © OASIS Open 2013. All Rights Reserved.

16 Cloud Service Archive (CSAR)

This section defines the metadata of a cloud service archive as well as its overall structure.

16.1 Overall Structure of a CSAR

A CSAR is a zip file containing at least two directories, the *TOSCA-Metadata* directory and the *Definitions* directory. Beyond that, other directories MAY be contained in a CSAR, i.e. the creator of a CSAR has all freedom to define the content of a CSAR and the structuring of this content as appropriate for the cloud application.

The TOSCA-Metadata directory contains metadata describing the other content of the CSAR. This metadata is referred to as *TOSCA meta file*. This file is named `TOSCA` and has the file extension `.meta`.

The Definitions directory contains one or more TOSCA Definitions documents (file extension `.tosca`). These Definitions files typically contain definitions related to the cloud application of the CSAR. In addition, CSARs can contain just the definition of elements for re-use in other contexts. For example, a CSAR might be used to package a set of Node Types and Relationship Types with their respective implementations that can then be used by Service Templates provided in other CSARs. In cases where a complete cloud application is packaged in a CSAR, one of the Definitions documents in the Definitions directory MUST contain a Service Template definition that defines the structure and behavior of the cloud application.

16.2 TOSCA Meta File

The TOSCA meta file includes metadata that allows interpreting the various artifacts within the CSAR properly. The `TOSCA.meta` file is contained in the `TOSCA-Metadata` directory of the CSAR.

A TOSCA meta file consists of name/value pairs. The name-part of a name/value pair is followed by a colon, followed by a blank, followed by the value-part of the name/value pair. The name MUST NOT contain a colon. Values that represent binary data MUST be base64 encoded. Values that extend beyond one line can be spread over multiple lines if each subsequent line starts with at least one space. Such spaces are then collapsed when the value string is read.

```
01 <name>: <value>
```

Each name/value pair is in a separate line. A list of related name/value pairs, i.e. a list of consecutive name/value pairs describing a particular file in a CSAR, is called a *block*. Blocks are separated by an empty line. The first block, called *block_0*, is metadata about the CSAR itself. All other blocks represent metadata of files in the CSAR.

The structure of block_0 in the TOSCA meta file is as follows:

```
01 TOSCA-Meta-File-Version: digit.digit
02 CSAR-Version: digit.digit
03 Created-By: string
04 Entry-Definitions: string ?
```

The name/value pairs are as follows:

- `TOSCA-Meta-File-Version`: This is the version number of the TOSCA meta file format. The value MUST be “1.0” in the current version of the TOSCA specification.
- `CSAR-Version`: This is the version number of the CSAR specification. The value MUST be “1.0” in the current version of the TOSCA specification.
- `Created-By`: The person or vendor, respectively, who created the CSAR.

 Copyright © OASIS Open 2013. All Rights Reserved.

- `Entry-Definitions`: This OPTIONAL name/value pair references a TOSCA Definitions file from the Definitions directory of the CSAR that SHOULD be used as entry point for processing the contents of the CSAR.
  Note, that a CSAR may contain multiple Definitions files. One reason for this is completeness, e.g. a Service Template defined in one of the Definitions files could refer to Node Types defined in another Definitions file that might be included in the Definitions directory to avoid importing it from external locations. The `Entry-Definitions` name/value pair is a hint to allow optimized processing of the set of files in the Definitions directory.

The first line of a block (other than block_0) MUST be a name/value pair that has the name "Name" and the value of which is the path-name of the file described. The second line MUST be a name/value pair that has the name "Content-Type" describing the type of the file described; the format is that of a MIME type with type/subtype structure. The other name/value pairs that consecutively follow are file-type specific.

```
01 Name: <path-name_1>
02 Content-Type: type_1/subtype_1
03 <name_11>: <value_11>
04 <name_12>: <value_12>
05 ...
06 <name_1n>: <value_1n>
07
08 ...
09
10 Name: <path-name_k>
11 Content-Type: type_k/subtype_k
12 <name_k1>: <value_k1>
13 <name_k2>: <value_k2>
14 ...
15 <name_km>: <value_km>
```

The name/value pairs are as follows:

- `Name`: The pathname or pathname pattern of the file(s) or resources described within the actual CSAR.
  Note, that the file located at this location MAY basically contain a reference to an external file. Such a reference is given by a URI that is of one of the URL schemes "file", "http", or "https".
- `Content-Type`: The type of the file described. This type is a MIME type complying with the type/subtype structure. Vendor defined subtypes SHOULD start as usual with the string "vnd.".

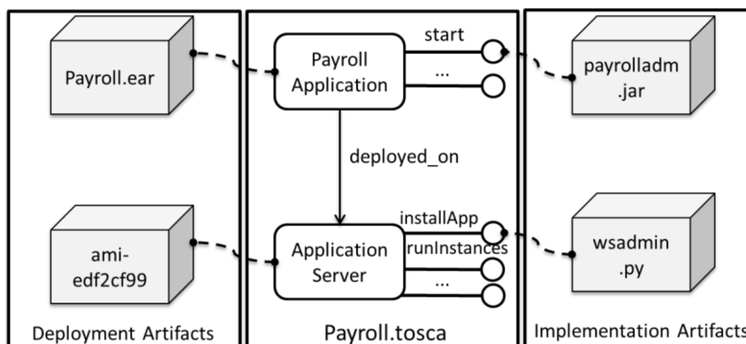
Note that later directives override earlier directives. This allows for specifying global default directives that can be specialized by later directorives in the TOSCA meta file.

16.3 Example

Figure 7 depicts a sample Definitions file named `Payroll.tosca` containing a Service Template of an application. The application is a payroll application written in Java that MUST be deployed on a proper application server. The Service Template of the application defines the Node Template `Payroll Application`, the Node Template `Application Server`, as well as the Relationship Template `deployed_on`. The `Payroll Application` is associated with an EAR file (named `Payroll.ear`) which is provided as corresponding Deployment Artifact of the `Payroll Application` Node Template. An Amazon Machine Image (AMI) is the Deployment Artifact of the `Application Server` Node Template; this Deployment Artifact is a reference to the image in the Amazon EC2 environment. The Implementation Artifacts of some operations of the Node Templates are

 Copyright © OASIS Open 2013. All Rights Reserved.

3016 provided too; for example, the `start` operation of the `Payroll Application` is implemented by a
3017 Java API supported by the `payrolladm.jar` file, the `installApp` operation of the `Application`
3018 `Server` is realized by the Python script `wsadmin.py`, while the `runInstances` operation is a REST
3019 API available at Amazon for running instances of an AMI. Note, that the `runInstances` operation is
3020 not related to a particular implementation artifact because it is available as an Amazon Web Service
3021 (https://ec2.amazonaws.com/?Action=RunInstances); but the details of this REST API are specified with
3022 the operation of the `Application Server` Node Type.

3023 

3024 Figure 7: Sample Service Template

3025 The corresponding Node Types and Relationship Types have been defined in the
3026 `PayrollTypes.tosca` document, which is imported by the Definitions document containing the
3027 `Payroll` Service Template. The following listing provides some of the details:

```
3028 01 <Definitions id="PayrollDefinitions"
3029 02              targetNamespace="http://www.example.com/tosca"
3030 03              xmlns:pay="http://www.example.com/tosca/Types">
3031 04
3032 05   <Import namespace="http://www.example.com/tosca/Types"
3033 06           location="http://www.example.com/tosca/Types/PayrollTypes.tosca"
3034 07           importType=" http://docs.oasis-open.org/tosca/ns/2011/12"/>
3035 08
3036 09   <Types>
3037 10      ...
3038 11   </Types>
3039 12
3040 13   <ServiceTemplate id="Payroll" name="Payroll Service Template">
3041 14
3042 15     <TopologyTemplate ID="PayrollTemplate">
3043 16
3044 17       <NodeTemplate id="Payroll Application"
3045 18                     type="pay:ApplicationNodeType">
3046 19         ...
3047 20
3048 21         <DeploymentArtifacts>
3049 22           <DeploymentArtifact name="PayrollEAR"
3050 23                               type="http://www.example.com/
3051 24                                     ns/tosca/2011/12/
3052 25                                     DeploymentArtifactTypes/CSARref">
3053 26                 EARs/Payroll.ear
3054 27           </DeploymentArtifact>
3055 28         </DeploymentArtifacts>
3056 29
3057 30       </NodeTemplate>
3058 31
3059 32       <NodeTemplate id="Application Server"
3060 33                     type="pay:ApplicationServerNodeType">
```

 Copyright © OASIS Open 2013. All Rights Reserved.

3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084

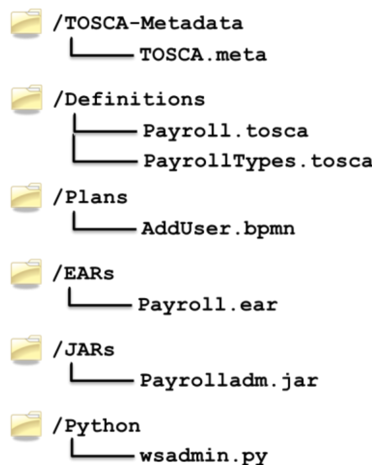
```
34          ...
35
36          <DeploymentArtifacts>
37            <DeploymentArtifact name="ApplicationServerImage"
38                                type="http://www.example.com/
39                                      ns/tosca/2011/12/
40                                      DeploymentArtifactTypes/AMIref">
41                   ami-edf2cf99
42            </DeploymentArtifact>
43          </DeploymentArtifacts>
44
45        </NodeTemplate>
46
47        <RelationshipTemplate id="deployed_on"
48                              type="pay:deployed_on">
49            <SourceElement ref="Payroll Application"/>
50            <TargetElement ref="Application Server"/>
51        </RelationshipTemplate>
52
53      </TopologyTemplate>
54
55    </ServiceTemplate>
56
57 </Definitions>
```

3085

3086 The Payroll Application Node Template specifies the deployment artifact PayrollEAR. It is a
3087 reference to the CSAR containing the Payroll.tosca file, which is indicated by the .../CSARref
3088 type of the DeploymentArtifact element. The type specific content is a path expression in the
3089 directory structure of the CSAR: it points to the Payroll.ear file in the EARs directory of the CSAR
3090 (see Figure 8 for the structure of the corresponding CSAR).

3091 The Application Server Node Template has a DeploymentArtifact called
3092 ApplicationServerImage that is a reference to an AMI (Amazon Machine Image), indicated by an
3093 .../AMIref type.

3094 The corresponding CSAR has the following structure (see Figure 8): The TOSCA.meta file is contained
3095 in the TOSCA-Metadata directory. The Payroll.tosca file itself is contained in the Service-
3096 Template directory. Also, the PayrollTypes.tosca file is in this directory. The content of the other
3097 directories has been sketched before.



3098
3099 Figure 8: Structure of CSAR Sample

 Copyright © OASIS Open 2013. All Rights Reserved.

3100 The `TOSCA.meta` file is as follows:

```
3101 01 TOSCA-Meta-Version: 1.0
3102 02 CSAR-Version: 1.0
3103 03 Created-By: Frank
3104 04
3105 05 Name: Service-Template/Payroll.tosca
3106 06 Content-Type: application/vnd.oasis.tosca.definitions
3107 07
3108 08 Name: Service-Template/PayrollTypes.tosca
3109 09 Content-Type: application/vnd.oasis.tosca.definitions
3110 10
3111 11 Name: Plans/AddUser.bpmn
3112 12 Content-Type: application/vnd.oasis.bpmn
3113 13
3114 14 Name: EARs/Payroll.ear
3115 15 Content-Type: application/vnd.oasis.ear
3116 16
3117 17 Name: JARs/Payrolladm.jar
3118 18 Content-Type: application/vnd.oasis.jar
3119 19
3120 20 Name: Python/wsadmin.py
3121 21 Content-Type: application/vnd.oasis.py
```

3122

 Copyright © OASIS Open 2013. All Rights Reserved.

3123 # 17 Security Considerations

3124 TOSCA does not mandate the use of any specific mechanism or technology for client authentication.
3125 However, a client MUST provide a principal or the principal MUST be obtainable by the infrastructure.

 Copyright © OASIS Open 2013. All Rights Reserved.

18 Conformance

A TOSCA Definitions document conforms to this specification if it conforms to the TOSCA schema and follows the syntax and semantics defined in the normative portions of this specification. The TOSCA schema takes precedence over the TOSCA grammar (pseudo schema as defined in section 2.5), which in turn takes precedence over normative text, which in turn takes precedence over examples.

An implementation conforms to this specification if it can process a conformant TOSCA Definitions document according to the rules described in chapters 4 through 16 of this specification.

This specification allows extensions. Each implementation SHALL fully support all required functionality of the specification exactly as specified. The use of extensions SHALL NOT contradict nor cause the non-conformance of functionality defined in the specification.

 Copyright © OASIS Open 2013. All Rights Reserved.

3136 # Appendix A. Portability and Interoperability
3137 Considerations

3138 This section illustrates the portability and interoperability aspects addressed by Service Templates:

3139 Portability - The ability to take Service Templates created in one vendor's environment and use them in
3140 another vendor's environment.

3141 Interoperability - The capability for multiple components (e.g. a task of a plan and the definition of a
3142 topology node) to interact using well-defined messages and protocols. This enables combining
3143 components from different vendors allowing seamless management of services.

3144 Portability demands support of TOSCA elements.

 Copyright © OASIS Open 2013. All Rights Reserved.

3145 # Appendix B. Acknowledgements

3146 The following individuals have participated in the creation of this specification and are gratefully
3147 acknowledged.

3148 **Participants:**

Aaron Zhang	Huawei Technologies Co., Ltd.
Adolf Hohl	NetApp
Afkham Azeez	WSO2
Al DeLucca	IBM
Alex Heneveld	Cloudsoft Corporation Limited
Allen Bannon	SAP AG
Anthony Rutkowski	Yaana Technologies, LLC
Arvind Srinivasan	IBM
Bryan Haynie	VCE
Bryan Murray	Hewlett-Packard
Chandrasekhar Sundaresh	CA Technologies
Charith Wickramarachchi	WSO2
Colin Hopkinson	3M HIS
Dale Moberg	Axway Software
Debojyoti Dutta	Cisco Systems
Dee Schur	OASIS
Denis Nothern	CenturyLink
Denis Weerasiri	WSO2
Derek Palma	Vnomic
Dhiraj Pathak	PricewaterhouseCoopers LLP:
Diane Mueller	ActiveState Software, Inc.
Doug Davis	IBM
Douglas Neuse	CA Technologies
Duncan Johnston-Watt	Cloudsoft Corporation Limited
Efraim Moscovich	CA Technologies
Frank Leymann	IBM
Gerd Breiter	IBM
James Thomason	Gale Technologies
Jan Ignatius	Nokia Siemens Networks GmbH & Co. KG
Jie Zhu	Huawei Technologies Co., Ltd.
John Wilmes	Individual
Joseph Malek	VCE
Ken Zink	CA Technologies
Kevin Poulter	SAP AG
Kevin Wilson	Hewlett-Packard
Koert Struijk	CA Technologies
Lee Thompson	Morphlabs, Inc.
li peng	Huawei Technologies Co., Ltd.
Marvin Waschke	CA Technologies
Mascot Yu	Huawei Technologies Co., Ltd.
Matthew Dovey	JISC Executive, University of Bristol
Matthew Rutkowski	IBM
Michael Schuster	SAP AG
Mike Edwards	IBM

 Copyright © OASIS Open 2013. All Rights Reserved.

Naveen Joy	Cisco Systems
Nikki Heron	rPath, Inc.
Paul Fremantle	WSO2
Paul Lipton	CA Technologies
Paul Zhang	Huawei Technologies Co., Ltd.
Rachid Sijelmassi	CA Technologies
Ravi Akireddy	Cisco Systems
Richard Bill	Jericho Systems
Richard Probst	SAP AG
Robert Evans	Zenoss, Inc.
Roland Wartenberg	Citrix Systems
Satoshi Konno	Morphlabs, Inc.
Sean Shen	China Internet Network Information Center(CNNIC)
Selvaratnam Uthaiyashankar	WSO2
Senaka Fernando	WSO2
Sherry Yu	Red Hat
Shumin Cheng	Huawei Technologies Co., Ltd.
Simon Moser	IBM
Srinath Perera	WSO2
Stephen Tyler	CA Technologies
Steve Fanshier	Software AG, Inc.
Steve Jones	Capgemini
Steve Winkler	SAP AG
Tad Deffler	CA Technologies
Ted Streete	VCE
Thilina Buddhika	WSO2
Thomas Spatzier	IBM
Tobias Kunze	Red Hat
Wang Xuan	Primeton Technologies, Inc.
wayne adams	EMC
Wenbo Zhu	Google Inc.
Xiaonan Song	Primeton Technologies, Inc.
YanJiong WANG	Primeton Technologies, Inc.
Zhexuan Song	Huawei Technologies Co., Ltd.

3149

Copyright © OASIS Open 2013. All Rights Reserved.

3150 # Appendix C. Complete TOSCA Grammar

3151 **Note:** The following is a pseudo EBNF grammar notation meant for documentation purposes only. The
3152 grammar is not intended for machine processing.

```
3153 01 <Definitions id="xs:ID"
3154 02              name="xs:string"?
3155 03              targetNamespace="xs:anyURI">
3156 04
3157 05   <Extensions>
3158 06     <Extension namespace="xs:anyURI"
3159 07                mustUnderstand="yes|no"?/> +
3160 08   </Extensions> ?
3161 09
3162 10   <Import namespace="xs:anyURI"?
3163 11           location="xs:anyURI"?
3164 12           importType="xs:anyURI"/> *
3165 13
3166 14   <Types>
3167 15     <xs:schema .../> *
3168 16   </Types> ?
3169 17
3170 18   (
3171 19     <ServiceTemplate id="xs:ID"
3172 20                      name="xs:string"?
3173 21                      targetNamespace="xs:anyURI"
3174 22                      substitutableNodeType="xs:QName"?>
3175 23
3176 24       <Tags>
3177 25         <Tag name="xs:string" value="xs:string"/> +
3178 26       </Tags> ?
3179 27
3180 28       <BoundaryDefinitions>
3181 29         <Properties>
3182 30           XML fragment
3183 31           <PropertyMappings>
3184 32             <PropertyMapping serviceTemplatePropertyRef="xs:string"
3185 33                              targetObjectRef="xs:IDREF"
3186 34                              targetPropertyRef="xs:IDREF"/> +
3187 35           </PropertyMappings/> ?
3188 36         </Properties> ?
3189 37
3190 38         <PropertyConstraints>
3191 39           <PropertyConstraint property="xs:string"
3192 40                               constraintType="xs:anyURI"> +
3193 41             constraint ?
3194 42           </PropertyConstraint>
3195 43         </PropertyConstraints> ?
3196 44
3197 45         <Requirements>
3198 46           <Requirement name="xs:string" ref="xs:IDREF"/> +
3199 47         </Requirements> ?
3200 48
3201 49         <Capabilities>
3202 50           <Capability name="xs:string" ref="xs:IDREF"/> +
3203 51         </Capabilities> ?
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
52
53         <Policies>
54           <Policy name="xs:string"? policyType="xs:QName"
55                   policyRef="xs:QName"?>
56             policy specific content ?
57           </Policy> +
58         </Policies> ?
59
60         <Interfaces>
61           <Interface name="xs:NCName">
62             <Operation name="xs:NCName">
63               (
64                 <NodeOperation nodeRef="xs:IDREF"
65                                interfaceName="xs:anyURI"
66                                operationName="xs:NCName"/>
67               |
68                 <RelationshipOperation relationshipRef="xs:IDREF"
69                                        interfaceName="xs:anyURI"
70                                        operationName="xs:NCName"/>
71               |
72                 <Plan planRef="xs:IDREF"/>
73               )
74             </Operation> +
75           </Interface> +
76         </Interfaces> ?
77
78       </BoundaryDefinitions> ?
79
80       <TopologyTemplate>
81         (
82           <NodeTemplate id="xs:ID" name="xs:string"? type="xs:QName"
83                         minInstances="xs:integer"?
84                         maxInstances="xs:integer | xs:string"?>
85             <Properties>
86               XML fragment
87             </Properties> ?
88
89             <PropertyConstraints>
90               <PropertyConstraint property="xs:string"
91                                   constraintType="xs:anyURI">
92                 constraint ?
93               </PropertyConstraint> +
94             </PropertyConstraints> ?
95
96             <Requirements>
97               <Requirement id="xs:ID" name="xs:string" type="xs:QName"> +
98                 <Properties>
99                   XML fragment
100                  <Properties> ?
101                  <PropertyConstraints>
102                    <PropertyConstraint property="xs:string"
103                                        constraintType="xs:anyURI"> +
104                      constraint ?
105                    </PropertyConstraint>
106                  </PropertyConstraints> ?
107                </Requirement>
108              </Requirements> ?
109
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
110               <Capabilities>
111                 <Capability id="xs:ID" name="xs:string"
112                             type="xs:QName"> +
113                   <Properties>
114                     XML fragment
115                   <Properties> ?
116                   <PropertyConstraints>
117                     <PropertyConstraint property="xs:string"
118                                         constraintType="xs:anyURI">
119                       constraint ?
120                     </PropertyConstraint> +
121                   </PropertyConstraints> ?
122                 </Capability>
123               </Capabilities> ?
124
125               <Policies>
126                 <Policy name="xs:string"? policyType="xs:QName"
127                         policyRef="xs:QName"?>
128                   policy specific content ?
129                 </Policy> +
130               </Policies> ?
131
132               <DeploymentArtifacts>
133                 <DeploymentArtifact name="xs:string"
134                                     artifactType="xs:QName"
135                                     artifactRef="xs:QName"?>
136                   artifact specific content ?
137                 </DeploymentArtifact> +
138               </DeploymentArtifacts> ?
139             </NodeTemplate>
140           |
141             <RelationshipTemplate id="xs:ID" name="xs:string"?
142                                   type="xs:QName">
143               <Properties>
144                 XML fragment
145               </Properties> ?
146
147               <PropertyConstraints>
148                 <PropertyConstraint property="xs:string"
149                                     constraintType="xs:anyURI">
150                   constraint ?
151                 </PropertyConstraint> +
152               </PropertyConstraints> ?
153
154               <SourceElement ref="xs:IDREF"/>
155               <TargetElement ref="xs:IDREF"/>
156
157               <RelationshipConstraints>
158                 <RelationshipConstraint constraintType="xs:anyURI">
159                   constraint ?
160                 </RelationshipConstraint> +
161               </RelationshipConstraints> ?
162
163             </RelationshipTemplate>
164           ) +
165         </TopologyTemplate>
166
167         <Plans>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3320 168           <Plan id="xs:ID"
3321 169                 name="xs:string"?
3322 170                 planType="xs:anyURI"
3323 171                 planLanguage="xs:anyURI">
3324 172
3325 173              <Precondition expressionLanguage="xs:anyURI">
3326 174                condition
3327 175              </Precondition> ?
3328 176
3329 177              <InputParameters>
3330 178                <InputParameter name="xs:string" type="xs:string"
3331 179                                required="yes|no"?/> +
3332 180              </InputParameters> ?
3333 181
3334 182              <OutputParameters>
3335 183                <OutputParameter name="xs:string" type="xs:string"
3336 184                                 required="yes|no"?/> +
3337 185              </OutputParameters> ?
3338 186
3339 187             (
3340 188               <PlanModel>
3341 189                 actual plan
3342 190               </PlanModel>
3343 191             |
3344 192               <PlanModelReference reference="xs:anyURI"/>
3345 193             )
3346 194
3347 195           </Plan> +
3348 196         </Plans> ?
3349 197
3350 198       </ServiceTemplate>
3351 199     |
3352 200       <NodeType name="xs:NCName" targetNamespace="xs:anyURI"?
3353 201                 abstract="yes|no"? final="yes|no"?>
3354 202
3355 203         <DerivedFrom typeRef="xs:QName"/> ?
3356 204
3357 205         <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3358 206
3359 207         <RequirementDefinitions>
3360 208           <RequirementDefinition name="xs:string"
3361 209                                  requirementType="xs:QName"
3362 210                                  lowerBound="xs:integer"?
3363 211                                  upperBound="xs:integer | xs:string"?>
3364 212             <Constraints>
3365 213               <Constraint constraintType="xs:anyURI">
3366 214                 constraint type specific content
3367 215               </Constraint> +
3368 216             </Constraints> ?
3369 217           </RequirementDefinition> +
3370 218         </RequirementDefinitions> ?
3371 219
3372 220         <CapabilityDefinitions>
3373 221           <CapabilityDefinition name="xs:string"
3374 222                                 capabilityType="xs:QName"
3375 223                                 lowerBound="xs:integer"?
3376 224                                 upperBound="xs:integer | xs:string"?>
3377 225             <Constraints>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3378 226               <Constraint constraintType="xs:anyURI">
3379 227                 constraint type specific content
3380 228               </Constraint> +
3381 229             </Constraints> ?
3382 230           </CapabilityDefinition> +
3383 231         </CapabilityDefinitions>
3384 232
3385 233         <InstanceStates>
3386 234           <InstanceState state="xs:anyURI"> +
3387 235         </InstanceStates> ?
3388 236
3389 237         <Interfaces>
3390 238           <Interface name="xs:NCName | xs:anyURI">
3391 239             <Operation name="xs:NCName">
3392 240               <InputParameters>
3393 241                 <InputParameter name="xs:string" type="xs:string"
3394 242                                 required="yes|no"?/> +
3395 243               </InputParameters> ?
3396 244               <OutputParameters>
3397 245                 <OutputParameter name="xs:string" type="xs:string"
3398 246                                  required="yes|no"?/> +
3399 247               </OutputParameters> ?
3400 248             </Operation> +
3401 249           </Interface> +
3402 250         </Interfaces> ?
3403 251
3404 252       </NodeType>
3405 253     |
3406 254       <NodeTypeImplementation name="xs:NCName"
3407 255                               targetNamespace="xs:anyURI"?
3408 256                               nodeType="xs:QName"
3409 257                               abstract="yes|no"?
3410 258                               final="yes|no"?>
3411 259
3412 260         <DerivedFrom nodeTypeImplementationRef="xs:QName"/> ?
3413 261
3414 262         <RequiredContainerFeatures>
3415 263           <RequiredContainerFeature feature="xs:anyURI"/> +
3416 264         </RequiredContainerFeatures> ?
3417 265
3418 266         <ImplementationArtifacts>
3419 267           <ImplementationArtifact interfaceName="xs:NCName | xs:anyURI"?
3420 268                                   operationName="xs:NCName"?
3421 269                                   artifactType="xs:QName"
3422 270                                   artifactRef="xs:QName"?>
3423 271             artifact specific content ?
3424 272           <ImplementationArtifact> +
3425 273         </ImplementationArtifacts> ?
3426 274
3427 275         <DeploymentArtifacts>
3428 276           <DeploymentArtifact name="xs:string" artifactType="xs:QName"
3429 277                               artifactRef="xs:QName"?>
3430 278             artifact specific content ?
3431 279           <DeploymentArtifact> +
3432 280         </DeploymentArtifacts> ?
3433 281
3434 282       </NodeTypeImplementation>
3435 283     |
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3436 284      <RelationshipType name="xs:NCName"
3437 285                        targetNamespace="xs:anyURI"?
3438 286                        abstract="yes|no"?
3439 287                        final="yes|no"?> +
3440 288
3441 289        <DerivedFrom typeRef="xs:QName"/> ?
3442 290
3443 291        <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3444 292
3445 293        <InstanceStates>
3446 294          <InstanceState state="xs:anyURI"> +
3447 295        </InstanceStates> ?
3448 296
3449 297        <SourceInterfaces>
3450 298          <Interface name="xs:NCName | xs:anyURI">
3451 299            <Operation name="xs:NCName">
3452 300              <InputParameters>
3453 301                <InputParameter name="xs:string" type="xs:string"
3454 302                                required="yes|no"?/> +
3455 303              </InputParameters> ?
3456 304              <OutputParameters>
3457 305                <OutputParameter name="xs:string" type="xs:string"
3458 306                                 required="yes|no"?/> +
3459 307              </OutputParameters> ?
3460 308            </Operation> +
3461 309          </Interface> +
3462 310        </SourceInterfaces> ?
3463 311
3464 312        <TargetInterfaces>
3465 313          <Interface name="xs:NCName | xs:anyURI">
3466 314            <Operation name="xs:NCName">
3467 315              <InputParameters>
3468 316                <InputParameter name="xs:string" type="xs:string"
3469 317                                required="yes|no"?/> +
3470 318              </InputParameters> ?
3471 319              <OutputParameters>
3472 320                <OutputParameter name="xs:string" type="xs:string"
3473 321                                 required="yes|no"?/> +
3474 322              </OutputParameters> ?
3475 323            </Operation> +
3476 324          </Interface> +
3477 325        </TargetInterfaces> ?
3478 326
3479 327        <ValidSource typeRef="xs:QName"/> ?
3480 328
3481 329        <ValidTarget typeRef="xs:QName"/> ?
3482 330
3483 331      </RelationshipType>
3484 332    |
3485 333      <RelationshipTypeImplementation name="xs:NCName"
3486 334                                      targetNamespace="xs:anyURI"?
3487 335                                      relationshipType="xs:QName"
3488 336                                      abstract="yes|no"?
3489 337                                      final="yes|no"?>
3490 338
3491 339        <DerivedFrom relationshipTypeImplementationRef="xs:QName"/> ?
3492 340
3493 341        <RequiredContainerFeatures>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3494 342              <RequiredContainerFeature feature="xs:anyURI"/> +
3495 343            </RequiredContainerFeatures> ?
3496 344
3497 345            <ImplementationArtifacts>
3498 346              <ImplementationArtifact interfaceName="xs:NCName | xs:anyURI"?
3499 347                                      operationName="xs:NCName"?
3500 348                                      artifactType="xs:QName"
3501 349                                      artifactRef="xs:QName"?>
3502 350                artifact specific content ?
3503 351              <ImplementationArtifact> +
3504 352            </ImplementationArtifacts> ?
3505 353
3506 354          </RelationshipTypeImplementation>
3507 355        |
3508 356          <RequirementType name="xs:NCName"
3509 357                           targetNamespace="xs:anyURI"?
3510 358                           abstract="yes|no"?
3511 359                           final="yes|no"?
3512 360                           requiredCapabilityType="xs:QName"?>
3513 361
3514 362            <DerivedFrom typeRef="xs:QName"/> ?
3515 363
3516 364            <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3517 365
3518 366          </RequirementType>
3519 367        |
3520 368          <CapabilityType name="xs:NCName"
3521 369                          targetNamespace="xs:anyURI"?
3522 370                          abstract="yes|no"?
3523 371                          final="yes|no"?>
3524 372
3525 373            <DerivedFrom typeRef="xs:QName"/> ?
3526 374
3527 375            <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3528 376
3529 377          </CapabilityType>
3530 378        |
3531 379          <ArtifactType name="xs:NCName"
3532 380                        targetNamespace="xs:anyURI"?
3533 381                        abstract="yes|no"?
3534 382                        final="yes|no"?>
3535 383
3536 384            <DerivedFrom typeRef="xs:QName"/> ?
3537 385
3538 386            <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3539 387
3540 388          </ArtifactType>
3541 389        |
3542 390          <ArtifactTemplate id="xs:ID" name="xs:string"? type="xs:QName">
3543 391
3544 392            <Properties>
3545 393              XML fragment
3546 394            </Properties> ?
3547 395
3548 396            <PropertyConstraints>
3549 397              <PropertyConstraint property="xs:string"
3550 398                                  constraintType="xs:anyURI"> +
3551 399                constraint ?
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3552 400            </PropertyConstraint>
3553 401          </PropertyConstraints> ?
3554 402
3555 403          <ArifactReferences>
3556 404            <ArtifactReference reference="xs:anyURI">
3557 405              (
3558 406               <Include pattern="xs:string"/>
3559 407              |
3560 408               <Exclude pattern="xs:string"/>
3561 409              )*
3562 410            </ArtifactReference> +
3563 411          </ArtifactReferences> ?
3564 412
3565 413        </ArtifactTemplate>
3566 414      |
3567 415        <PolicyType name="xs:NCName"
3568 416                    policyLanguage="xs:anyURI"?
3569 417                    abstract="yes|no"?
3570 418                    final="yes|no"?
3571 419                    targetNamespace="xs:anyURI"?>
3572 420          <Tags>
3573 421            <Tag name="xs:string" value="xs:string"/> +
3574 422          </Tags> ?
3575 423
3576 424          <DerivedFrom typeRef="xs:QName"/> ?
3577 425
3578 426          <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
3579 427
3580 428          <AppliesTo>
3581 429            <NodeTypeReference typeRef="xs:QName"/> +
3582 430          </AppliesTo> ?
3583 431
3584 432          policy type specific content ?
3585 433
3586 434        </PolicyType>
3587 435      |
3588 436        <PolicyTemplate id="xs:ID" name="xs:string"? type="xs:QName">
3589 437
3590 438          <Properties>
3591 439            XML fragment
3592 440          </Properties> ?
3593 441
3594 442          <PropertyConstraints>
3595 443            <PropertyConstraint property="xs:string"
3596 444                                constraintType="xs:anyURI"> +
3597 445              constraint ?
3598 446            </PropertyConstraint>
3599 447          </PropertyConstraints> ?
3600 448
3601 449          policy type specific content ?
3602 450
3603 451        </PolicyTemplate>
3604 452      ) +
3605 453
3606 454    </Definitions>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

Appendix D. TOSCA Schema

TOSCA-v1.0.xsd:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <xs:schema targetNamespace="http://docs.oasis-open.org/tosca/ns/2011/12"
03  elementFormDefault="qualified" attributeFormDefault="unqualified"
04  xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
05  xmlns:xs="http://www.w3.org/2001/XMLSchema">
06
07  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
08   schemaLocation="http://www.w3.org/2001/xml.xsd"/>
09
10  <xs:element name="documentation" type="tDocumentation"/>
11  <xs:complexType name="tDocumentation" mixed="true">
12   <xs:sequence>
13    <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
14   </xs:sequence>
15   <xs:attribute name="source" type="xs:anyURI"/>
16   <xs:attribute ref="xml:lang"/>
17  </xs:complexType>
18
19  <xs:complexType name="tExtensibleElements">
20   <xs:sequence>
21    <xs:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
22    <xs:any namespace="##other" processContents="lax" minOccurs="0"
23     maxOccurs="unbounded"/>
24   </xs:sequence>
25   <xs:anyAttribute namespace="##other" processContents="lax"/>
26  </xs:complexType>
27
28  <xs:complexType name="tImport">
29   <xs:complexContent>
30    <xs:extension base="tExtensibleElements">
31     <xs:attribute name="namespace" type="xs:anyURI"/>
32     <xs:attribute name="location" type="xs:anyURI"/>
33     <xs:attribute name="importType" type="importedURI" use="required"/>
34    </xs:extension>
35   </xs:complexContent>
36  </xs:complexType>
37
38  <xs:element name="Definitions">
39   <xs:complexType>
40    <xs:complexContent>
41     <xs:extension base="tDefinitions"/>
42    </xs:complexContent>
43   </xs:complexType>
44  </xs:element>
45  <xs:complexType name="tDefinitions">
46   <xs:complexContent>
47    <xs:extension base="tExtensibleElements">
48     <xs:sequence>
49      <xs:element name="Extensions" minOccurs="0">
50       <xs:complexType>
51        <xs:sequence>
52         <xs:element name="Extension" type="tExtension"
```


Copyright © OASIS Open 2013. All Rights Reserved.

```
3661  53          maxOccurs="unbounded"/>
3662  54        </xs:sequence>
3663  55       </xs:complexType>
3664  56      </xs:element>
3665  57      <xs:element name="Import" type="tImport" minOccurs="0"
3666  58       maxOccurs="unbounded"/>
3667  59      <xs:element name="Types" minOccurs="0">
3668  60       <xs:complexType>
3669  61        <xs:sequence>
3670  62         <xs:any namespace="##other" processContents="lax" minOccurs="0"
3671  63          maxOccurs="unbounded"/>
3672  64        </xs:sequence>
3673  65       </xs:complexType>
3674  66      </xs:element>
3675  67      <xs:choice maxOccurs="unbounded">
3676  68       <xs:element name="ServiceTemplate" type="tServiceTemplate"/>
3677  69       <xs:element name="NodeType" type="tNodeType"/>
3678  70       <xs:element name="NodeTypeImplementation"
3679  71        type="tNodeTypeImplementation"/>
3680  72       <xs:element name="RelationshipType" type="tRelationshipType"/>
3681  73       <xs:element name="RelationshipTypeImplementation"
3682  74        type="tRelationshipTypeImplementation"/>
3683  75       <xs:element name="RequirementType" type="tRequirementType"/>
3684  76       <xs:element name="CapabilityType" type="tCapabilityType"/>
3685  77       <xs:element name="ArtifactType" type="tArtifactType"/>
3686  78       <xs:element name="ArtifactTemplate" type="tArtifactTemplate"/>
3687  79       <xs:element name="PolicyType" type="tPolicyType"/>
3688  80       <xs:element name="PolicyTemplate" type="tPolicyTemplate"/>
3689  81      </xs:choice>
3690  82     </xs:sequence>
3691  83     <xs:attribute name="id" type="xs:ID" use="required"/>
3692  84     <xs:attribute name="name" type="xs:string" use="optional"/>
3693  85     <xs:attribute name="targetNamespace" type="xs:anyURI" use="required"/>
3694  86    </xs:extension>
3695  87   </xs:complexContent>
3696  88  </xs:complexType>
3697  89
3698  90  <xs:complexType name="tServiceTemplate">
3699  91   <xs:complexContent>
3700  92    <xs:extension base="tExtensibleElements">
3701  93     <xs:sequence>
3702  94      <xs:element name="Tags" type="tTags" minOccurs="0"/>
3703  95      <xs:element name="BoundaryDefinitions" type="tBoundaryDefinitions"
3704  96       minOccurs="0"/>
3705  97      <xs:element name="TopologyTemplate" type="tTopologyTemplate"/>
3706  98      <xs:element name="Plans" type="tPlans" minOccurs="0"/>
3707  99     </xs:sequence>
3708  100       <xs:attribute name="id" type="xs:ID" use="required"/>
3709  101       <xs:attribute name="name" type="xs:string" use="optional"/>
3710  102       <xs:attribute name="targetNamespace" type="xs:anyURI"/>
3711  103       <xs:attribute name="substitutableNodeType" type="xs:QName"
3712  104        use="optional"/>
3713  105      </xs:extension>
3714  106     </xs:complexContent>
3715  107    </xs:complexType>
3716  108
3717  109    <xs:complexType name="tTags">
3718  110     <xs:sequence>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3719 111      <xs:element name="Tag" type="tTag" maxOccurs="unbounded"/>
3720 112     </xs:sequence>
3721 113   </xs:complexType>
3722 114
3723 115   <xs:complexType name="tTag">
3724 116    <xs:attribute name="name" type="xs:string" use="required"/>
3725 117    <xs:attribute name="value" type="xs:string" use="required"/>
3726 118   </xs:complexType>
3727 119
3728 120   <xs:complexType name="tBoundaryDefinitions">
3729 121    <xs:sequence>
3730 122     <xs:element name="Properties" minOccurs="0">
3731 123      <xs:complexType>
3732 124       <xs:sequence>
3733 125        <xs:any namespace="##other"/>
3734 126        <xs:element name="PropertyMappings" minOccurs="0">
3735 127         <xs:complexType>
3736 128          <xs:sequence>
3737 129           <xs:element name="PropertyMapping" type="tPropertyMapping"
3738 130            maxOccurs="unbounded"/>
3739 131          </xs:sequence>
3740 132         </xs:complexType>
3741 133        </xs:element>
3742 134       </xs:sequence>
3743 135      </xs:complexType>
3744 136     </xs:element>
3745 137     <xs:element name="PropertyConstraints" minOccurs="0">
3746 138      <xs:complexType>
3747 139       <xs:sequence>
3748 140        <xs:element name="PropertyConstraint" type="tPropertyConstraint"
3749 141         maxOccurs="unbounded"/>
3750 142       </xs:sequence>
3751 143      </xs:complexType>
3752 144     </xs:element>
3753 145     <xs:element name="Requirements" minOccurs="0">
3754 146      <xs:complexType>
3755 147       <xs:sequence>
3756 148        <xs:element name="Requirement" type="tRequirementRef"
3757 149         maxOccurs="unbounded"/>
3758 150       </xs:sequence>
3759 151      </xs:complexType>
3760 152     </xs:element>
3761 153     <xs:element name="Capabilities" minOccurs="0">
3762 154      <xs:complexType>
3763 155       <xs:sequence>
3764 156        <xs:element name="Capability" type="tCapabilityRef"
3765 157         maxOccurs="unbounded"/>
3766 158       </xs:sequence>
3767 159      </xs:complexType>
3768 160     </xs:element>
3769 161     <xs:element name="Policies" minOccurs="0">
3770 162      <xs:complexType>
3771 163       <xs:sequence>
3772 164        <xs:element name="Policy" type="tPolicy" maxOccurs="unbounded"/>
3773 165       </xs:sequence>
3774 166      </xs:complexType>
3775 167     </xs:element>
3776 168     <xs:element name="Interfaces" minOccurs="0">
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3777 169       <xs:complexType>
3778 170        <xs:sequence>
3779 171         <xs:element name="Interface" type="tExportedInterface"
3780 172          maxOccurs="unbounded"/>
3781 173        </xs:sequence>
3782 174       </xs:complexType>
3783 175      </xs:element>
3784 176     </xs:sequence>
3785 177    </xs:complexType>
3786 178
3787 179    <xs:complexType name="tPropertyMapping">
3788 180     <xs:attribute name="serviceTemplatePropertyRef" type="xs:string"
3789 181      use="required"/>
3790 182     <xs:attribute name="targetObjectRef" type="xs:IDREF" use="required"/>
3791 183     <xs:attribute name="targetPropertyRef" type="xs:string"
3792 184      use="required"/>
3793 185    </xs:complexType>
3794 186
3795 187    <xs:complexType name="tRequirementRef">
3796 188     <xs:attribute name="name" type="xs:string" use="optional"/>
3797 189     <xs:attribute name="ref" type="xs:IDREF" use="required"/>
3798 190    </xs:complexType>
3799 191
3800 192    <xs:complexType name="tCapabilityRef">
3801 193     <xs:attribute name="name" type="xs:string" use="optional"/>
3802 194     <xs:attribute name="ref" type="xs:IDREF" use="required"/>
3803 195    </xs:complexType>
3804 196
3805 197    <xs:complexType name="tEntityType" abstract="true">
3806 198     <xs:complexContent>
3807 199      <xs:extension base="tExtensibleElements">
3808 200       <xs:sequence>
3809 201        <xs:element name="Tags" type="tTags" minOccurs="0"/>
3810 202        <xs:element name="DerivedFrom" minOccurs="0">
3811 203         <xs:complexType>
3812 204          <xs:attribute name="typeRef" type="xs:QName" use="required"/>
3813 205         </xs:complexType>
3814 206        </xs:element>
3815 207        <xs:element name="PropertiesDefinition" minOccurs="0">
3816 208         <xs:complexType>
3817 209          <xs:attribute name="element" type="xs:QName"/>
3818 210          <xs:attribute name="type" type="xs:QName"/>
3819 211         </xs:complexType>
3820 212        </xs:element>
3821 213       </xs:sequence>
3822 214       <xs:attribute name="name" type="xs:NCName" use="required"/>
3823 215       <xs:attribute name="abstract" type="tBoolean" default="no"/>
3824 216       <xs:attribute name="final" type="tBoolean" default="no"/>
3825 217       <xs:attribute name="targetNamespace" type="xs:anyURI"
3826 218        use="optional"/>
3827 219      </xs:extension>
3828 220     </xs:complexContent>
3829 221    </xs:complexType>
3830 222
3831 223    <xs:complexType name="tEntityTemplate" abstract="true">
3832 224     <xs:complexContent>
3833 225      <xs:extension base="tExtensibleElements">
3834 226       <xs:sequence>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3835 227        <xs:element name="Properties" minOccurs="0">
3836 228         <xs:complexType>
3837 229          <xs:sequence>
3838 230           <xs:any namespace="##other" processContents="lax"/>
3839 231          </xs:sequence>
3840 232         </xs:complexType>
3841 233        </xs:element>
3842 234        <xs:element name="PropertyConstraints" minOccurs="0">
3843 235         <xs:complexType>
3844 236          <xs:sequence>
3845 237           <xs:element name="PropertyConstraint"
3846 238            type="tPropertyConstraint" maxOccurs="unbounded"/>
3847 239          </xs:sequence>
3848 240         </xs:complexType>
3849 241        </xs:element>
3850 242       </xs:sequence>
3851 243       <xs:attribute name="id" type="xs:ID" use="required"/>
3852 244       <xs:attribute name="type" type="xs:QName" use="required"/>
3853 245      </xs:extension>
3854 246     </xs:complexContent>
3855 247    </xs:complexType>
3856 248
3857 249    <xs:complexType name="tNodeTemplate">
3858 250     <xs:complexContent>
3859 251      <xs:extension base="tEntityTemplate">
3860 252       <xs:sequence>
3861 253        <xs:element name="Requirements" minOccurs="0">
3862 254         <xs:complexType>
3863 255          <xs:sequence>
3864 256           <xs:element name="Requirement" type="tRequirement"
3865 257            maxOccurs="unbounded"/>
3866 258          </xs:sequence>
3867 259         </xs:complexType>
3868 260        </xs:element>
3869 261        <xs:element name="Capabilities" minOccurs="0">
3870 262         <xs:complexType>
3871 263          <xs:sequence>
3872 264           <xs:element name="Capability" type="tCapability"
3873 265            maxOccurs="unbounded"/>
3874 266          </xs:sequence>
3875 267         </xs:complexType>
3876 268        </xs:element>
3877 269        <xs:element name="Policies" minOccurs="0">
3878 270         <xs:complexType>
3879 271          <xs:sequence>
3880 272           <xs:element name="Policy" type="tPolicy"
3881 273            maxOccurs="unbounded"/>
3882 274          </xs:sequence>
3883 275         </xs:complexType>
3884 276        </xs:element>
3885 277        <xs:element name="DeploymentArtifacts" type="tDeploymentArtifacts"
3886 278         minOccurs="0"/>
3887 279       </xs:sequence>
3888 280       <xs:attribute name="name" type="xs:string" use="optional"/>
3889 281       <xs:attribute name="minInstances" type="xs:int" use="optional"
3890 282        default="1"/>
3891 283       <xs:attribute name="maxInstances" use="optional" default="1">
3892 284        <xs:simpleType>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3893 285         <xs:union>
3894 286          <xs:simpleType>
3895 287           <xs:restriction base="xs:nonNegativeInteger">
3896 288            <xs:pattern value="([1-9]+[0-9]*)"/>
3897 289           </xs:restriction>
3898 290          </xs:simpleType>
3899 291          <xs:simpleType>
3900 292           <xs:restriction base="xs:string">
3901 293            <xs:enumeration value="unbounded"/>
3902 294           </xs:restriction>
3903 295          </xs:simpleType>
3904 296         </xs:union>
3905 297        </xs:simpleType>
3906 298       </xs:attribute>
3907 299      </xs:extension>
3908 300     </xs:complexContent>
3909 301    </xs:complexType>
3910 302
3911 303    <xs:complexType name="tTopologyTemplate">
3912 304     <xs:complexContent>
3913 305      <xs:extension base="tExtensibleElements">
3914 306       <xs:choice maxOccurs="unbounded">
3915 307        <xs:element name="NodeTemplate" type="tNodeTemplate"/>
3916 308        <xs:element name="RelationshipTemplate"
3917 309         type="tRelationshipTemplate"/>
3918 310       </xs:choice>
3919 311      </xs:extension>
3920 312     </xs:complexContent>
3921 313    </xs:complexType>
3922 314
3923 315    <xs:complexType name="tRelationshipType">
3924 316     <xs:complexContent>
3925 317      <xs:extension base="tEntityType">
3926 318       <xs:sequence>
3927 319        <xs:element name="InstanceStates"
3928 320         type="tTopologyElementInstanceStates" minOccurs="0"/>
3929 321        <xs:element name="SourceInterfaces" minOccurs="0">
3930 322         <xs:complexType>
3931 323          <xs:sequence>
3932 324           <xs:element name="Interface" type="tInterface"
3933 325            maxOccurs="unbounded"/>
3934 326          </xs:sequence>
3935 327         </xs:complexType>
3936 328        </xs:element>
3937 329        <xs:element name="TargetInterfaces" minOccurs="0">
3938 330         <xs:complexType>
3939 331          <xs:sequence>
3940 332           <xs:element name="Interface" type="tInterface"
3941 333            maxOccurs="unbounded"/>
3942 334          </xs:sequence>
3943 335         </xs:complexType>
3944 336        </xs:element>
3945 337        <xs:element name="ValidSource" minOccurs="0">
3946 338         <xs:complexType>
3947 339          <xs:attribute name="typeRef" type="xs:QName" use="required"/>
3948 340         </xs:complexType>
3949 341        </xs:element>
3950 342        <xs:element name="ValidTarget" minOccurs="0">
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
3951 343         <xs:complexType>
3952 344          <xs:attribute name="typeRef" type="xs:QName" use="required"/>
3953 345         </xs:complexType>
3954 346        </xs:element>
3955 347       </xs:sequence>
3956 348      </xs:extension>
3957 349     </xs:complexContent>
3958 350    </xs:complexType>
3959 351
3960 352    <xs:complexType name="tRelationshipTypeImplementation">
3961 353     <xs:complexContent>
3962 354      <xs:extension base="tExtensibleElements">
3963 355       <xs:sequence>
3964 356        <xs:element name="Tags" type="tTags" minOccurs="0"/>
3965 357        <xs:element name="DerivedFrom" minOccurs="0">
3966 358         <xs:complexType>
3967 359          <xs:attribute name="relationshipTypeImplementationRef"
3968 360           type="xs:QName" use="required"/>
3969 361         </xs:complexType>
3970 362        </xs:element>
3971 363        <xs:element name="RequiredContainerFeatures"
3972 364         type="tRequiredContainerFeatures" minOccurs="0"/>
3973 365        <xs:element name="ImplementationArtifacts"
3974 366         type="tImplementationArtifacts" minOccurs="0"/>
3975 367       </xs:sequence>
3976 368       <xs:attribute name="name" type="xs:NCName" use="required"/>
3977 369       <xs:attribute name="targetNamespace" type="xs:anyURI"
3978 370        use="optional"/>
3979 371       <xs:attribute name="relationshipType" type="xs:QName"
3980 372        use="required"/>
3981 373       <xs:attribute name="abstract" type="tBoolean" use="optional"
3982 374        default="no"/>
3983 375       <xs:attribute name="final" type="tBoolean" use="optional"
3984 376        default="no"/>
3985 377      </xs:extension>
3986 378     </xs:complexContent>
3987 379    </xs:complexType>
3988 380
3989 381    <xs:complexType name="tRelationshipTemplate">
3990 382     <xs:complexContent>
3991 383      <xs:extension base="tEntityTemplate">
3992 384       <xs:sequence>
3993 385        <xs:element name="SourceElement">
3994 386         <xs:complexType>
3995 387          <xs:attribute name="ref" type="xs:IDREF" use="required"/>
3996 388         </xs:complexType>
3997 389        </xs:element>
3998 390        <xs:element name="TargetElement">
3999 391         <xs:complexType>
4000 392          <xs:attribute name="ref" type="xs:IDREF" use="required"/>
4001 393         </xs:complexType>
4002 394        </xs:element>
4003 395        <xs:element name="RelationshipConstraints" minOccurs="0">
4004 396         <xs:complexType>
4005 397          <xs:sequence>
4006 398           <xs:element name="RelationshipConstraint"
4007 399            maxOccurs="unbounded">
4008 400            <xs:complexType>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4009 401               <xs:sequence>
4010 402                <xs:any namespace="##other" processContents="lax"
4011 403                 minOccurs="0"/>
4012 404               </xs:sequence>
4013 405               <xs:attribute name="constraintType" type="xs:anyURI"
4014 406                use="required"/>
4015 407              </xs:complexType>
4016 408             </xs:element>
4017 409            </xs:sequence>
4018 410           </xs:complexType>
4019 411          </xs:element>
4020 412         </xs:sequence>
4021 413         <xs:attribute name="name" type="xs:string" use="optional"/>
4022 414        </xs:extension>
4023 415       </xs:complexContent>
4024 416      </xs:complexType>
4025 417
4026 418      <xs:complexType name="tNodeType">
4027 419       <xs:complexContent>
4028 420        <xs:extension base="tEntityType">
4029 421         <xs:sequence>
4030 422          <xs:element name="RequirementDefinitions" minOccurs="0">
4031 423           <xs:complexType>
4032 424            <xs:sequence>
4033 425             <xs:element name="RequirementDefinition"
4034 426              type="tRequirementDefinition" maxOccurs="unbounded"/>
4035 427            </xs:sequence>
4036 428           </xs:complexType>
4037 429          </xs:element>
4038 430          <xs:element name="CapabilityDefinitions" minOccurs="0">
4039 431           <xs:complexType>
4040 432            <xs:sequence>
4041 433             <xs:element name="CapabilityDefinition"
4042 434              type="tCapabilityDefinition" maxOccurs="unbounded"/>
4043 435            </xs:sequence>
4044 436           </xs:complexType>
4045 437          </xs:element>
4046 438          <xs:element name="InstanceStates"
4047 439           type="tTopologyElementInstanceStates" minOccurs="0"/>
4048 440          <xs:element name="Interfaces" minOccurs="0">
4049 441           <xs:complexType>
4050 442            <xs:sequence>
4051 443             <xs:element name="Interface" type="tInterface"
4052 444              maxOccurs="unbounded"/>
4053 445            </xs:sequence>
4054 446           </xs:complexType>
4055 447          </xs:element>
4056 448         </xs:sequence>
4057 449        </xs:extension>
4058 450       </xs:complexContent>
4059 451      </xs:complexType>
4060 452
4061 453      <xs:complexType name="tNodeTypeImplementation">
4062 454       <xs:complexContent>
4063 455        <xs:extension base="tExtensibleElements">
4064 456         <xs:sequence>
4065 457          <xs:element name="Tags" type="tTags" minOccurs="0"/>
4066 458          <xs:element name="DerivedFrom" minOccurs="0">
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4067 459        <xs:complexType>
4068 460         <xs:attribute name="nodeTypeImplementationRef" type="xs:QName"
4069 461          use="required"/>
4070 462        </xs:complexType>
4071 463       </xs:element>
4072 464       <xs:element name="RequiredContainerFeatures"
4073 465        type="tRequiredContainerFeatures" minOccurs="0"/>
4074 466       <xs:element name="ImplementationArtifacts"
4075 467        type="tImplementationArtifacts" minOccurs="0"/>
4076 468       <xs:element name="DeploymentArtifacts" type="tDeploymentArtifacts"
4077 469        minOccurs="0"/>
4078 470      </xs:sequence>
4079 471      <xs:attribute name="name" type="xs:NCName" use="required"/>
4080 472      <xs:attribute name="targetNamespace" type="xs:anyURI"
4081 473       use="optional"/>
4082 474      <xs:attribute name="nodeType" type="xs:QName" use="required"/>
4083 475      <xs:attribute name="abstract" type="tBoolean" use="optional"
4084 476       default="no"/>
4085 477      <xs:attribute name="final" type="tBoolean" use="optional"
4086 478       default="no"/>
4087 479     </xs:extension>
4088 480    </xs:complexContent>
4089 481   </xs:complexType>
4090 482
4091 483   <xs:complexType name="tRequirementType">
4092 484    <xs:complexContent>
4093 485     <xs:extension base="tEntityType">
4094 486      <xs:attribute name="requiredCapabilityType" type="xs:QName"
4095 487       use="optional"/>
4096 488     </xs:extension>
4097 489    </xs:complexContent>
4098 490   </xs:complexType>
4099 491
4100 492   <xs:complexType name="tRequirementDefinition">
4101 493    <xs:complexContent>
4102 494     <xs:extension base="tExtensibleElements">
4103 495      <xs:sequence>
4104 496       <xs:element name="Constraints" minOccurs="0">
4105 497        <xs:complexType>
4106 498         <xs:sequence>
4107 499          <xs:element name="Constraint" type="tConstraint"
4108 500           maxOccurs="unbounded"/>
4109 501         </xs:sequence>
4110 502        </xs:complexType>
4111 503       </xs:element>
4112 504      </xs:sequence>
4113 505      <xs:attribute name="name" type="xs:string" use="required"/>
4114 506      <xs:attribute name="requirementType" type="xs:QName"
4115 507       use="required"/>
4116 508      <xs:attribute name="lowerBound" type="xs:int" use="optional"
4117 509       default="1"/>
4118 510      <xs:attribute name="upperBound" use="optional" default="1">
4119 511       <xs:simpleType>
4120 512        <xs:union>
4121 513         <xs:simpleType>
4122 514          <xs:restriction base="xs:nonNegativeInteger">
4123 515           <xs:pattern value="([1-9]+[0-9]*)"/>
4124 516          </xs:restriction>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4125  517            </xs:simpleType>
4126  518            <xs:simpleType>
4127  519             <xs:restriction base="xs:string">
4128  520              <xs:enumeration value="unbounded"/>
4129  521             </xs:restriction>
4130  522            </xs:simpleType>
4131  523           </xs:union>
4132  524          </xs:simpleType>
4133  525         </xs:attribute>
4134  526        </xs:extension>
4135  527       </xs:complexContent>
4136  528      </xs:complexType>
4137  529
4138  530      <xs:complexType name="tRequirement">
4139  531       <xs:complexContent>
4140  532        <xs:extension base="tEntityTemplate">
4141  533         <xs:attribute name="name" type="xs:string" use="required"/>
4142  534        </xs:extension>
4143  535       </xs:complexContent>
4144  536      </xs:complexType>
4145  537
4146  538      <xs:complexType name="tCapabilityType">
4147  539       <xs:complexContent>
4148  540        <xs:extension base="tEntityType"/>
4149  541       </xs:complexContent>
4150  542      </xs:complexType>
4151  543
4152  544      <xs:complexType name="tCapabilityDefinition">
4153  545       <xs:complexContent>
4154  546        <xs:extension base="tExtensibleElements">
4155  547         <xs:sequence>
4156  548          <xs:element name="Constraints" minOccurs="0">
4157  549           <xs:complexType>
4158  550            <xs:sequence>
4159  551             <xs:element name="Constraint" type="tConstraint"
4160  552              maxOccurs="unbounded"/>
4161  553            </xs:sequence>
4162  554           </xs:complexType>
4163  555          </xs:element>
4164  556         </xs:sequence>
4165  557         <xs:attribute name="name" type="xs:string" use="required"/>
4166  558         <xs:attribute name="capabilityType" type="xs:QName"
4167  559          use="required"/>
4168  560         <xs:attribute name="lowerBound" type="xs:int" use="optional"
4169  561          default="1"/>
4170  562         <xs:attribute name="upperBound" use="optional" default="1">
4171  563          <xs:simpleType>
4172  564           <xs:union>
4173  565            <xs:simpleType>
4174  566             <xs:restriction base="xs:nonNegativeInteger">
4175  567              <xs:pattern value="([1-9]+[0-9]*)"/>
4176  568             </xs:restriction>
4177  569            </xs:simpleType>
4178  570            <xs:simpleType>
4179  571             <xs:restriction base="xs:string">
4180  572              <xs:enumeration value="unbounded"/>
4181  573             </xs:restriction>
4182  574            </xs:simpleType>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
575         </xs:union>
576        </xs:simpleType>
577       </xs:attribute>
578      </xs:extension>
579     </xs:complexContent>
580    </xs:complexType>
581
582    <xs:complexType name="tCapability">
583     <xs:complexContent>
584      <xs:extension base="tEntityTemplate">
585       <xs:attribute name="name" type="xs:string" use="required"/>
586      </xs:extension>
587     </xs:complexContent>
588    </xs:complexType>
589
590    <xs:complexType name="tArtifactType">
591     <xs:complexContent>
592      <xs:extension base="tEntityType"/>
593     </xs:complexContent>
594    </xs:complexType>
595
596    <xs:complexType name="tArtifactTemplate">
597     <xs:complexContent>
598      <xs:extension base="tEntityTemplate">
599       <xs:sequence>
600        <xs:element name="ArtifactReferences" minOccurs="0">
601         <xs:complexType>
602          <xs:sequence>
603           <xs:element name="ArtifactReference" type="tArtifactReference"
604            maxOccurs="unbounded"/>
605          </xs:sequence>
606         </xs:complexType>
607        </xs:element>
608       </xs:sequence>
609       <xs:attribute name="name" type="xs:string" use="optional"/>
610      </xs:extension>
611     </xs:complexContent>
612    </xs:complexType>
613
614    <xs:complexType name="tDeploymentArtifacts">
615     <xs:sequence>
616      <xs:element name="DeploymentArtifact" type="tDeploymentArtifact"
617       maxOccurs="unbounded"/>
618     </xs:sequence>
619    </xs:complexType>
620
621    <xs:complexType name="tDeploymentArtifact">
622     <xs:complexContent>
623      <xs:extension base="tExtensibleElements">
624       <xs:attribute name="name" type="xs:string" use="required"/>
625       <xs:attribute name="artifactType" type="xs:QName" use="required"/>
626       <xs:attribute name="artifactRef" type="xs:QName" use="optional"/>
627      </xs:extension>
628     </xs:complexContent>
629    </xs:complexType>
630
631    <xs:complexType name="tImplementationArtifacts">
632     <xs:sequence>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4241 633     <xs:element name="ImplementationArtifact" maxOccurs="unbounded">
4242 634      <xs:complexType>
4243 635       <xs:complexContent>
4244 636        <xs:extension base="tImplementationArtifact"/>
4245 637       </xs:complexContent>
4246 638      </xs:complexType>
4247 639     </xs:element>
4248 640    </xs:sequence>
4249 641   </xs:complexType>
4250 642
4251 643   <xs:complexType name="tImplementationArtifact">
4252 644    <xs:complexContent>
4253 645     <xs:extension base="tExtensibleElements">
4254 646      <xs:attribute name="interfaceName" type="xs:anyURI"
4255 647       use="optional"/>
4256 648      <xs:attribute name="operationName" type="xs:NCName"
4257 649       use="optional"/>
4258 650      <xs:attribute name="artifactType" type="xs:QName" use="required"/>
4259 651      <xs:attribute name="artifactRef" type="xs:QName" use="optional"/>
4260 652     </xs:extension>
4261 653    </xs:complexContent>
4262 654   </xs:complexType>
4263 655
4264 656   <xs:complexType name="tPlans">
4265 657    <xs:sequence>
4266 658     <xs:element name="Plan" type="tPlan" maxOccurs="unbounded"/>
4267 659    </xs:sequence>
4268 660    <xs:attribute name="targetNamespace" type="xs:anyURI"
4269 661     use="optional"/>
4270 662   </xs:complexType>
4271 663
4272 664   <xs:complexType name="tPlan">
4273 665    <xs:complexContent>
4274 666     <xs:extension base="tExtensibleElements">
4275 667      <xs:sequence>
4276 668       <xs:element name="Precondition" type="tCondition" minOccurs="0"/>
4277 669       <xs:element name="InputParameters" minOccurs="0">
4278 670        <xs:complexType>
4279 671         <xs:sequence>
4280 672          <xs:element name="InputParameter" type="tParameter"
4281 673           maxOccurs="unbounded"/>
4282 674         </xs:sequence>
4283 675        </xs:complexType>
4284 676       </xs:element>
4285 677       <xs:element name="OutputParameters" minOccurs="0">
4286 678        <xs:complexType>
4287 679         <xs:sequence>
4288 680          <xs:element name="OutputParameter" type="tParameter"
4289 681           maxOccurs="unbounded"/>
4290 682         </xs:sequence>
4291 683        </xs:complexType>
4292 684       </xs:element>
4293 685       <xs:choice>
4294 686        <xs:element name="PlanModel">
4295 687         <xs:complexType>
4296 688          <xs:sequence>
4297 689           <xs:any namespace="##other" processContents="lax"/>
4298 690          </xs:sequence>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4299 691          </xs:complexType>
4300 692         </xs:element>
4301 693         <xs:element name="PlanModelReference">
4302 694          <xs:complexType>
4303 695           <xs:attribute name="reference" type="xs:anyURI"
4304 696            use="required"/>
4305 697          </xs:complexType>
4306 698         </xs:element>
4307 699        </xs:choice>
4308 700       </xs:sequence>
4309 701       <xs:attribute name="id" type="xs:ID" use="required"/>
4310 702       <xs:attribute name="name" type="xs:string" use="optional"/>
4311 703       <xs:attribute name="planType" type="xs:anyURI" use="required"/>
4312 704       <xs:attribute name="planLanguage" type="xs:anyURI" use="required"/>
4313 705      </xs:extension>
4314 706     </xs:complexContent>
4315 707    </xs:complexType>
4316 708
4317 709    <xs:complexType name="tPolicyType">
4318 710     <xs:complexContent>
4319 711      <xs:extension base="tEntityType">
4320 712       <xs:sequence>
4321 713        <xs:element name="AppliesTo" type="tAppliesTo" minOccurs="0"/>
4322 714       </xs:sequence>
4323 715       <xs:attribute name="policyLanguage" type="xs:anyURI"
4324 716        use="optional"/>
4325 717      </xs:extension>
4326 718     </xs:complexContent>
4327 719    </xs:complexType>
4328 720
4329 721    <xs:complexType name="tPolicyTemplate">
4330 722     <xs:complexContent>
4331 723      <xs:extension base="tEntityTemplate">
4332 724       <xs:attribute name="name" type="xs:string" use="optional"/>
4333 725      </xs:extension>
4334 726     </xs:complexContent>
4335 727    </xs:complexType>
4336 728
4337 729    <xs:complexType name="tAppliesTo">
4338 730     <xs:sequence>
4339 731      <xs:element name="NodeTypeReference" maxOccurs="unbounded">
4340 732       <xs:complexType>
4341 733        <xs:attribute name="typeRef" type="xs:QName" use="required"/>
4342 734       </xs:complexType>
4343 735      </xs:element>
4344 736     </xs:sequence>
4345 737    </xs:complexType>
4346 738
4347 739    <xs:complexType name="tPolicy">
4348 740     <xs:complexContent>
4349 741      <xs:extension base="tExtensibleElements">
4350 742       <xs:attribute name="name" type="xs:string" use="optional"/>
4351 743       <xs:attribute name="policyType" type="xs:QName" use="required"/>
4352 744       <xs:attribute name="policyRef" type="xs:QName" use="optional"/>
4353 745      </xs:extension>
4354 746     </xs:complexContent>
4355 747    </xs:complexType>
4356 748
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4357 749   <xs:complexType name="tConstraint">
4358 750    <xs:sequence>
4359 751     <xs:any namespace="##other" processContents="lax"/>
4360 752    </xs:sequence>
4361 753    <xs:attribute name="constraintType" type="xs:anyURI" use="required"/>
4362 754   </xs:complexType>
4363 755
4364 756   <xs:complexType name="tPropertyConstraint">
4365 757    <xs:complexContent>
4366 758     <xs:extension base="tConstraint">
4367 759      <xs:attribute name="property" type="xs:string" use="required"/>
4368 760     </xs:extension>
4369 761    </xs:complexContent>
4370 762   </xs:complexType>
4371 763
4372 764   <xs:complexType name="tExtensions">
4373 765    <xs:complexContent>
4374 766     <xs:extension base="tExtensibleElements">
4375 767      <xs:sequence>
4376 768       <xs:element name="Extension" type="tExtension"
4377 769        maxOccurs="unbounded"/>
4378 770      </xs:sequence>
4379 771     </xs:extension>
4380 772    </xs:complexContent>
4381 773   </xs:complexType>
4382 774
4383 775   <xs:complexType name="tExtension">
4384 776    <xs:complexContent>
4385 777     <xs:extension base="tExtensibleElements">
4386 778      <xs:attribute name="namespace" type="xs:anyURI" use="required"/>
4387 779      <xs:attribute name="mustUnderstand" type="tBoolean" use="optional"
4388 780       default="yes"/>
4389 781     </xs:extension>
4390 782    </xs:complexContent>
4391 783   </xs:complexType>
4392 784
4393 785   <xs:complexType name="tParameter">
4394 786    <xs:attribute name="name" type="xs:string" use="required"/>
4395 787    <xs:attribute name="type" type="xs:string" use="required"/>
4396 788    <xs:attribute name="required" type="tBoolean" use="optional"
4397 789     default="yes"/>
4398 790   </xs:complexType>
4399 791
4400 792   <xs:complexType name="tInterface">
4401 793    <xs:sequence>
4402 794     <xs:element name="Operation" type="tOperation"
4403 795      maxOccurs="unbounded"/>
4404 796    </xs:sequence>
4405 797    <xs:attribute name="name" type="xs:anyURI" use="required"/>
4406 798   </xs:complexType>
4407 799
4408 800   <xs:complexType name="tExportedInterface">
4409 801    <xs:sequence>
4410 802     <xs:element name="Operation" type="tExportedOperation"
4411 803      maxOccurs="unbounded"/>
4412 804    </xs:sequence>
4413 805    <xs:attribute name="name" type="xs:anyURI" use="required"/>
4414 806   </xs:complexType>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4415  807
4416  808    <xs:complexType name="tOperation">
4417  809     <xs:complexContent>
4418  810      <xs:extension base="tExtensibleElements">
4419  811       <xs:sequence>
4420  812        <xs:element name="InputParameters" minOccurs="0">
4421  813         <xs:complexType>
4422  814          <xs:sequence>
4423  815           <xs:element name="InputParameter" type="tParameter"
4424  816            maxOccurs="unbounded"/>
4425  817          </xs:sequence>
4426  818         </xs:complexType>
4427  819        </xs:element>
4428  820        <xs:element name="OutputParameters" minOccurs="0">
4429  821         <xs:complexType>
4430  822          <xs:sequence>
4431  823           <xs:element name="OutputParameter" type="tParameter"
4432  824            maxOccurs="unbounded"/>
4433  825          </xs:sequence>
4434  826         </xs:complexType>
4435  827        </xs:element>
4436  828       </xs:sequence>
4437  829       <xs:attribute name="name" type="xs:NCName" use="required"/>
4438  830      </xs:extension>
4439  831     </xs:complexContent>
4440  832    </xs:complexType>
4441  833
4442  834    <xs:complexType name="tExportedOperation">
4443  835     <xs:choice>
4444  836      <xs:element name="NodeOperation">
4445  837       <xs:complexType>
4446  838        <xs:attribute name="nodeRef" type="xs:IDREF" use="required"/>
4447  839        <xs:attribute name="interfaceName" type="xs:anyURI"
4448  840         use="required"/>
4449  841        <xs:attribute name="operationName" type="xs:NCName"
4450  842         use="required"/>
4451  843       </xs:complexType>
4452  844      </xs:element>
4453  845      <xs:element name="RelationshipOperation">
4454  846       <xs:complexType>
4455  847        <xs:attribute name="relationshipRef" type="xs:IDREF"
4456  848         use="required"/>
4457  849        <xs:attribute name="interfaceName" type="xs:anyURI"
4458  850         use="required"/>
4459  851        <xs:attribute name="operationName" type="xs:NCName"
4460  852         use="required"/>
4461  853       </xs:complexType>
4462  854      </xs:element>
4463  855      <xs:element name="Plan">
4464  856       <xs:complexType>
4465  857        <xs:attribute name="planRef" type="xs:IDREF" use="required"/>
4466  858       </xs:complexType>
4467  859      </xs:element>
4468  860     </xs:choice>
4469  861     <xs:attribute name="name" type="xs:NCName" use="required"/>
4470  862    </xs:complexType>
4471  863
4472  864    <xs:complexType name="tCondition">
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4473  865      <xs:sequence>
4474  866        <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
4475  867      </xs:sequence>
4476  868      <xs:attribute name="expressionLanguage" type="xs:anyURI"
4477  869        use="required"/>
4478  870    </xs:complexType>
4479  871
4480  872    <xs:complexType name="tTopologyElementInstanceStates">
4481  873      <xs:sequence>
4482  874        <xs:element name="InstanceState" maxOccurs="unbounded">
4483  875          <xs:complexType>
4484  876            <xs:attribute name="state" type="xs:anyURI" use="required"/>
4485  877          </xs:complexType>
4486  878        </xs:element>
4487  879      </xs:sequence>
4488  880    </xs:complexType>
4489  881
4490  882    <xs:complexType name="tArtifactReference">
4491  883      <xs:choice minOccurs="0" maxOccurs="unbounded">
4492  884        <xs:element name="Include">
4493  885          <xs:complexType>
4494  886            <xs:attribute name="pattern" type="xs:string" use="required"/>
4495  887          </xs:complexType>
4496  888        </xs:element>
4497  889        <xs:element name="Exclude">
4498  890          <xs:complexType>
4499  891            <xs:attribute name="pattern" type="xs:string" use="required"/>
4500  892          </xs:complexType>
4501  893        </xs:element>
4502  894      </xs:choice>
4503  895      <xs:attribute name="reference" type="xs:anyURI" use="required"/>
4504  896    </xs:complexType>
4505  897
4506  898    <xs:complexType name="tRequiredContainerFeatures">
4507  899      <xs:sequence>
4508  900        <xs:element name="RequiredContainerFeature"
4509  901          type="tRequiredContainerFeature" maxOccurs="unbounded"/>
4510  902      </xs:sequence>
4511  903    </xs:complexType>
4512  904
4513  905    <xs:complexType name="tRequiredContainerFeature">
4514  906      <xs:attribute name="feature" type="xs:anyURI" use="required"/>
4515  907    </xs:complexType>
4516  908
4517  909    <xs:simpleType name="tBoolean">
4518  910      <xs:restriction base="xs:string">
4519  911        <xs:enumeration value="yes"/>
4520  912        <xs:enumeration value="no"/>
4521  913      </xs:restriction>
4522  914    </xs:simpleType>
4523  915
4524  916    <xs:simpleType name="importedURI">
4525  917      <xs:restriction base="xs:anyURI"/>
4526  918    </xs:simpleType>
4527  919
4528  920  </xs:schema>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

Appendix E. Sample

This appendix contains the full sample used in this specification.

E.1 Sample Service Topology Definition

```
01 <Definitions name="MyServiceTemplateDefinition"
02              targetNamespace="http://www.example.com/sample">
03   <Types>
04     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
05                elementFormDefault="qualified"
06                attributeFormDefault="unqualified">
07       <xs:element name="ApplicationProperties">
08         <xs:complexType>
09           <xs:sequence>
10             <xs:element name="Owner" type="xs:string"/>
11             <xs:element name="InstanceName" type="xs:string"/>
12             <xs:element name="AccountID" type="xs:string"/>
13           </xs:sequence>
14         </xs:complexType>
15       </xs:element>
16       <xs:element name="AppServerProperties">
17         <xs:complexType>
18           <xs:sequence>
19             <element name="HostName" type="xs:string"/>
20             <element name="IPAddress" type="xs:string"/>
21             <element name="HeapSize" type="xs:positiveInteger"/>
22             <element name="SoapPort" type="xs:positiveInteger"/>
23           </xs:sequence>
24         </xs:complexType>
25       </xs:element>
26     </xs:schema>
27   </Types>
28
29   <ServiceTemplate id="MyServiceTemplate">
30
31     <Tags>
32       <Tag name="author" value="someone@example.com"/>
33     </Tags>
34
35     <TopologyTemplate id="SampleApplication">
36
37       <NodeTemplate id="MyApplication"
38                     name="My Application"
39                     nodeType="abc:Application">
40         <Properties>
41           <ApplicationProperties>
42             <Owner>Frank</Owner>
43             <InstanceName>Thomas' favorite application</InstanceName>
44           </ApplicationProperties>
45         </Properties>
46       </NodeTemplate>
47
48       <NodeTemplate id="MyAppServer"
49                     name="My Application Server"
```


Copyright © OASIS Open 2013. All Rights Reserved.

```
4581  50                      nodeType="abc:ApplicationServer"
4582  51                      minInstances="0"
4583  52                      maxInstances="unbounded"/>
4584  53
4585  54        <RelationshipTemplate id="MyDeploymentRelationship"
4586  55                              relationshipType="abc:deployedOn">
4587  56          <SourceElement id="MyApplication"/>
4588  57          <TargetElement id="MyAppServer"/>
4589  58        </RelationshipTemplate>
4590  59
4591  60      </TopologyTemplate>
4592  61
4593  62      <Plans>
4594  63        <Plan id="DeployApplication"
4595  64              name="Sample Application Build Plan"
4596  65              planType="http://docs.oasis-
4597  66               open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
4598  67              planLanguage="http://www.omg.org/spec/BPMN/20100524/MODEL">
4599  68
4600  69          <Precondition expressionLanguage="www.example.com/text"> ?
4601  70            Run only if funding is available
4602  71          </Precondition>
4603  72
4604  73          <PlanModel>
4605  74            <process name="DeployNewApplication" id="p1">
4606  75              <documentation>This process deploys a new instance of the
4607  76                sample application.
4608  77              </documentation>
4609  78
4610  79              <task id="t1" name="CreateAccount"/>
4611  80
4612  81              <task id="t2" name="AcquireNetworkAddresses"
4613  82                    isSequential="false"
4614  83                    loopDataInput="t2Input.LoopCounter"/>
4615  84                <documentation>Assumption: t2 gets data of type "input"
4616  85                  as input and this data has a field names "LoopCounter"
4617  86                  that contains the actual multiplicity of the task.
4618  87                </documentation>
4619  88
4620  89              <task id="t3" name="DeployApplicationServer"
4621  90                    isSequential="false"
4622  91                    loopDataInput="t3Input.LoopCounter"/>
4623  92
4624  93              <task id="t4" name="DeployApplication"
4625  94                    isSequential="false"
4626  95                    loopDataInput="t4Input.LoopCounter"/>
4627  96
4628  97              <sequenceFlow id="s1" targetRef="t2" sourceRef="t1"/>
4629  98              <sequenceFlow id="s2" targetRef="t3" sourceRef="t2"/>
4630  99              <sequenceFlow id="s3" targetRef="t4" sourceRef="t3"/>
4631  100            </process>
4632  101          </PlanModel>
4633  102        </Plan>
4634  103
4635  104        <Plan id="RemoveApplication"
4636  105              planType="http://docs.oasis-
4637  106               open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
4638  107              planLanguage="http://docs.oasis-
```

 Copyright © OASIS Open 2013. All Rights Reserved.

```
4639  108                open.org/wsbpel/2.0/process/executable">
4640  109            <PlanModelReference reference="prj:RemoveApp"/>
4641  110          </Plan>
4642  111        </Plans>
4643  112
4644  113      </ServiceTemplate>
4645  114
4646  115      <NodeType name="Application">
4647  116        <documentation xml:lang="EN">
4648  117          A reusable definition of a node type representing an
4649  118          application that can be deployed on application servers.
4650  119        </documentation>
4651  120        <NodeTypeProperties element="ApplicationProperties"/>
4652  121        <InstanceStates>
4653  122          <InstanceState state="http://www.example.com/started"/>
4654  123          <InstanceState state="http://www.example.com/stopped"/>
4655  124        </InstanceStates>
4656  125        <Interfaces>
4657  126          <Interface name="DeploymentInterface">
4658  127            <Operation name="DeployApplication">
4659  128              <InputParameters>
4660  129                <InputParamter name="InstanceName"
4661  130                               type="xs:string"/>
4662  131                <InputParamter name="AppServerHostname"
4663  132                               type="xs:string"/>
4664  133                <InputParamter name="ContextRoot"
4665  134                               type="xs:string"/>
4666  135              </InputParameters>
4667  136            </Operation>
4668  137          </Interface>
4669  138        </Interfaces
4670  139      </NodeType>
4671  140
4672  141      <NodeType name="ApplicationServer"
4673  142                targetNamespace="http://www.example.com/sample">
4674  143        <NodeTypeProperties element="AppServerProperties"/>
4675  144        <Interfaces>
4676  145          <Interface name="MyAppServerInterface">
4677  146            <Operation name="AcquireNetworkAddress"/>
4678  147            <Operation name="DeployApplicationServer"/>
4679  148          </Interface>
4680  149        </Interfaces>
4681  150      </NodeType>
4682  151
4683  152      <RelationshipType name="deployedOn">
4684  153        <documentation xml:lang="EN">
4685  154          A reusable definition of relation that expresses deployment of
4686  155          an artifact on a hosting environment.
4687  156        </documentation>
4688  157      </RelationshipType>
4689  158
4690  159    </Definitions>
```

 Copyright © OASIS Open 2013. All Rights Reserved.

4691

Appendix F. Revision History

Revision	Date	Editor	Changes Made
wd-01	2012-01-26	Thomas Spatzier	Changes for JIRA Issue TOSCA-1: Initial working draft based on input spec delivered to TOSCA TC. Copied all content from input spec and just changed namespace. Added line numbers to whole document.
wd-02	2012-02-23	Thomas Spatzier	Changes for JIRA Issue TOSCA-6: Reviewed and adapted normative statement keywords according to RFC2119.
wd-03	2012-03-06	Arvind Srinivasan, Thomas Spatzier	Changes for JIRA Issue TOSCA-10: Marked all occurrences of keywords from the TOSCA language (element and attribute names) in Courier New font.
wd-04	2012-03-22	Thomas Spatzier	Changes for JIRA Issue TOSCA-4: Changed definition of `NodeType Interfaces` element; adapted text and examples
wd-05	2012-03-30	Thomas Spatzier	Changes for JIRA Issue TOSCA-5: Changed definition of `NodeTemplate` to include `ImplementationArtifact` element; adapted text Added Acknowledgements section in Appendix
wd-06	2012-05-03	Thomas Spatzier, Derek Palma	Changes for JIRA Issue TOSCA-15: Added clarifying section about artifacts (see section 3.2); Implemented editorial changes according to OASIS staff recommendations; updated Acknowledgements section
wd-07	2012-06-15	Thomas Spatzier	Changes for JIRA Issue TOSCA-20: Added `abstract` attribute to NodeType for sub-issue 2; Added `final` attribute to NodeType for sub-issue 4; Added explanatory text on Node Type properties for sub-issue 8
wd-08	2012-06-29	Thomas Spatzier, Derek Palma	Changes for JIRA Issue TOSCA-23: Added interfaces and introduced inheritance for RelationshipType; based on wd-07 Added reference to XML element and attribute naming scheme used in this spec

Copyright © OASIS Open 2013. All Rights Reserved.

wd-09	2012-07-16	Thomas Spatzier	Changes for JIRA Issue TOSCA-17: Specifies the format of a CSAR file; Explained CSAR concept in the corresponding section.
wd-10	2012-07-30	Thomas Spatzier, Derek Palma	Changes for JIRA Issue TOSCA-18 and related issues: Introduced concept of Requirements and Capabilities; Restructuring of some paragraphs to improve readability
wd-11	2012-08-25	Thomas Spatzier, Derek Palma	Changes for JIRA Issue TOSCA-13: Clarifying rewording of introduction Changes for JIRA Issue TOSCA-38: Add `substituableNodeType` attribute and `BoundaryDefinitions` to Service Template to allow for Service Template composition. Changes for JIRA Issue TOSCA-41: Add Tags to Service Template as simple means for Service Template versioning; Changes for JIRA Issue TOSCA-47: Use `name` and `targetNamespace` for uniquely identifying TOSCA types; Changes for JIRA Issue TOSCA-48 (partly): implement notational conventions in pseudo schemas
wd-12	2012-09-29	Thomas Spatzier, Derek Palma	Editorial changes for TOSCA-10: Formatting corrections according to OASIS feedback Changes for JIRA Issue TOSCA-28,29: Added Node Type Implementation (with deployment artifacts and implementation artifacts) that points to a Node Type it realizes; added Relationship Type Implementation analogously for Relationship Types Changes for JIRA Issue TOSCA-38: Added `Interfaces` to `BoundaryDefinitions`. Changes for JIRA Issue TOSCA-52: Removal of `GroupTemplate` Changes for JIRA Issue TOSCA-54: Clarifying rewording in section 3.5 Changes for JIRA Issue TOSCA-56: Clarifying rewording in section 2.8.2 Changes for JIRA Issue TOSCA-58: Clarifying rewording in section 13 Updated roster as of 2012-09-29

Copyright © OASIS Open 2013. All Rights Reserved.

wd-13	2012-10-26	Thomas Spatzier, Derek Palma	Changes for JIRA Issue TOSCA-10: More fixes to formatting and references in document according to OASIS staff comments Changes for JIRA Issues TOSCA-36/37: Added `PolicyType` and `PolicyTemplate` elements to allow for re-usable definitions of policies. Changes for JIRA Issue TOSCA-57: Restructure TOSCA schema to allow for better modular definitions and separation of concerns. Changes for JIRA Issue TOSCA-59: Rewording to clarify overriding of deployment artifacts of Node Templates. Some additional minor changes in wording. Changes for JIRA Issue TOSCA-63: clarifying rewording
wd-14	2012-11-19	Thomas Spatzier	Changes for JIRA Issue TOSCA-76: Add Entry-Definitions property for TOSCA.meta file. Multiple general editorial fixes: Typos, namespaces and MIME types used in examples Fixed schema problems in tPolicyTemplate and tPolicyType Added text to Conformance section.
wd-15	2013-02-26	Thomas Spatzier	Changes for JIRA Issue TOSCA-79: Handle public review comments: fixes of typos and other non-material changes like inconsistencies between the specification document and the schema in this document and the TOSCA schema
wd-16	2013-04-15	Derek Palma, Thomas Spatzier	Changes for JIRA Issue TOSCA-82: Non-matierial change on namespace name use Changes for JIRA Issue TOSCA-83: fix broken references in document

4693

Copyright © OASIS Open 2013. All Rights Reserved.