TTA Standard

정보통신단체표준(국문표준)

TTAK.xx-xx.xxxx/R1 제정일: 20xx년 xx월 xx일

클라우드 애플리케이션을 위한 토폴로지 및 오케스트레이션 명세

Topology and Orchestration Specification for Cloud Applications(TOSCA)

표준초안 검토 위원회 클라우드 컴퓨팅 프로젝트그룹(PG1003)

표준안 심의 위원회 지능정보기반 기술위원회(TC010)

	성명	소 속	직위	위원회 및 직위	표준번호
표준(과제) 제안	허준호	TTA	책임연구원	PG1003 위원	
표준 초안 작성자	송현경	TTA	책임연구원	PG1003 위원	
	정신성	TTA	선임연구원	PG1003 위원	
	허준호	TTA	책임연구원	PG1003 위원	
	최현숙	TTA	팀장	PG1003 위원	
사무국 담당	박준환	TTA	선임연구원	-	

발행인 : 한국정보통신기술협회 회장
발행처 : 한국정보통신기술협회
13591, 경기도 성남시 분당구 분당로 47
Tel : 031-724-0114, Fax : 031-724-0109
발행일 : 20xx.xx

서 문

1 표준의 목적

이 표준은 클라우드 애플리케이션의 구조와 생명주기(배포, 확장 및 축소, 배포 해제 등)에 대한 명세를 XML 구문의 DSL(Domain Specific Language)로 정의하여 개발자가 이식성이 높은 클라우드 애플리케이션을 개발할 수 있도록 한다.

2 주요 내용 요약

이 표준은 OASIS의 Topology and Orchestration Specification for Cloud Applications(TOSCA) 표준에서 제시된 서비스 템플릿 개념을 사용하여 클라우드 애플리케이션의 구조와 생명주기를 정의한다. 서비스 템플릿은 클라우드 애플리케이션의 토폴로지 및 오케스트레이션을 정의하는데 사용되며, 이 표준은 서비스 템플릿을 정의하기 위한 노드, 관계, 기능, 아티팩트 등 TOSCA 유형과 정책 및 제약조건을 포함한다.

3 인용 표준과의 비교

3.1 인용 표준과의 관련성

이 표준은 OASIS의 “Topology and Orchestration Specification for Cloud Applications Version 1.0(2013)” 내용을 국문으로 동일하게 제공한다. 인용 표준에 대한 국문 번역 과정에서 의미상 차이점이 발생하는 경우 인용 표준을 따른다.

3.2 인용 표준과 본 표준의 비교표

TTAK.xx-xx.xxxx/R1	OASIS Topology and Orchestration Specification for Cloud Applications Version 1.0(2013)	비고
1 적용범위	-	추가
2 인용 표준	-	추가
3 용어 정의	-	추가
4 약어	-	추가
5 개요	1 Introduction	동일(번역)
6 언어 설계	2 Language Design	동일(번역)
7 핵심 개념 및 사용 패턴	3 Core Concepts and Usage Pattern	동일(번역)
8 TOSCA 정의서	4 The TOSCA Definitions Document	동일(번역)
9 서비스 템플릿	5 Service Templates	동일(번역)
10 노드 유형	6 Node Types	동일(번역)
11 노드 유형 구현	7 Node Type Implementations	동일(번역)
12 관계 유형	8 Relationship Types	동일(번역)
13 관계 유형 구현	9 Relationship Type Implementations	동일(번역)
14 요구사항 유형	10 Requirement Types	동일(번역)
15 기능 유형	11 Capability Types	동일(번역)
16 아티팩트 유형	12 Artifact Types	동일(번역)
17 아티팩트 템플릿	13 Artifact Templates	동일(번역)
18 정책 유형	14 Policy Types	동일(번역)
19 정책 템플릿	15 Policy Templates	동일(번역)
20 클라우드 서비스 아카이브(CSAR)	16 Cloud Service Archive(CSAR)	동일(번역)
21 보안 고려사항	17 Security Considerations	동일(번역)
-	18 Conformance	제외
-	Appendix A. Portability and Interoperability Considerations	제외
-	Appendix B. Acknowledgements	제외
-	Appendix C. Complete TOSCA Grammar	제외
-	Appendix D. TOSCA Schema	제외
-	Appendix E. Sample	제외
부록 I	-	추가

Preface

1 Purpose

The standard is to define meta model and life cycle(deployment, scaling out/in, undeployment etc.) of cloud applications in a DSL(Domain Specific Language) with XML syntax to implement portable and interoperable cloud applications.

2 Summary

The standard defines the structure and life cycle of a cloud applications using a service template concept presented in the OASIS Topology and Orchestration Specification for Cloud Applications(TOSCA) standard. The service template is used to define the topology and orchestration of cloud applications and this standard includes TOSCA types(e.g., Node, Relationship, Capability, Artifact, etc.), policies and constraints along with input or output declarations for defining the service template.

3 Relationship to Reference Standards

The standard provides the contents of “Topology and Orchestration Specification for Cloud Applications Version 1.0(2013)” in Korean. In the case where semantic differences exist between those standards, reference standard precedes this translated version.

목 차

클라우드 애플리케이션을 위한 토폴로지 및 오케스트레이션 명세 (Topology and Orchestration Specification for Cloud Applications(TOSCA))

1 적용 범위

본 표준은 클라우드 인프라나 플랫폼의 종류에 관계없이 클라우드 애플리케이션, 서비스, 및 기타 자원(예: 네트워크, 저장소, 가상머신, 컨테이너 등)의 관리자동화, 이식성 확보 위한 클라우드 애플리케이션 및 서비스의 토폴로지(구조)와 오케스트레이션 명세에 대한 것이다.

이를 위해, 클라우드 애플리케이션, 서비스 및 기타 자원에 대한 전체 생명주기(배포, 확장/축소, 배포 해제 등)의 오케스트레이션을 가능하게 하는 모델을 기계 및 사람이 읽을 수 있는 형식의 DSL(Domain Specific Language)로 정의한다. 즉, 이 모델을 통해 클라우드 애플리케이션 및 서비스의 구성 요소, 상호관계, 종속성, 요구사항 및 기능을 운영 정책의 문맥에 따라 DSL로 기술할 수 있다.

2 인용 표준

OASIS Topology and Orchestration Specification for Cloud Applications Version 1.0(2013)

3 용어 정의

해당 사항 없음

4 약어

TOSCA	Topology and Orchestration Specification for Cloud Applications
XML	Extensible Markup Language
ITIL	IT Infrastructure Library
OVF	Open Virtualization Format
EJB	Enterprise JavaBeans
REST	Representational State Transfer
CSAR	Cloud Service ARchive
MIME	Multipurpose Internet Mail Extensions

5 개요

애플리케이션 계층 서비스에 대한 생성 및 관리 자동화가 다른 클라우드 환경으로 이식될 수 있고 해당 애플리케이션 서비스들이 상호운용 가능하게 유지된다면, 클라우드 컴퓨팅의 가치는 더욱 빛날 것이다. 본 핵심 TOSCA 명세는 *서비스 토폴로지(Service Topology)*를 이용하여 서비스 컴포넌트와 그들의 관계를 서술하기 위한 언어를 제공하고 *오케스트레이션 프로세스(Orchestration Process)*를 이용하여 서비스를 생성하거나 변경하는 관리 절차를 서술하는 것을 가능하게 한다. *서비스 템플릿(Service Template)*에 토폴로지와 오케스트레이션을 조합하여 표현하며, 애플리케이션을 타 클라우드 환경으로 이식할 때, 상호운용 가능한 클라우드 서비스의 배포와 전체 수명주기(확장, 패치, 모니터링 등) 동안의 관리를 위해 보존해야 할 정보가 바로 여기에 들어간다.

6 언어 설계

TOSCA 언어는 *토폴로지 템플릿(Topology Template)*과 *계획(Plan)*을 통하여 서비스 템플릿을 서술하기 위한 문법을 정립하며 이는 설계 시점 형상(즉 교환 가능한 서비스 서술)에 중점을 둔다. 서비스 인스턴스(instance)를 관리하는 것을 지원하는 계획 모델을 명시하기 위한 컨테이너를 규정함으로써 실행 측면을 다룬다.

이 언어는 추가적인 공급 업체별 또는 분야별 정보로 기존 정의를 확장하는데 사용할 수 있는 확장 메커니즘을 제공한다.

6.1 타 명세에 대한 의존성

TOSCA는 다음 명세를 이용한다.

- XML schema 1.0

6.2 표기 규칙

이 문서에서 사용하는 키워드 “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, “OPTIONAL”과 같은 주요 표현은 [RFC2119]에서 서술한 바에 따라 해석한다. 이 명세는 [UNCEFACT XMLNDR]에서 기술된 XML 명명 및 설계 규칙을 준수한다. 즉, XML 요소 명에 대하여 대문자로 시작하는 카멜표기(camel-case)를 사용하고 XML 속성 명에 대하여 소문자로 시작하는 카멜표기(camel-case)를 사용한다.

6.3 규범적 참조(Normative References)

[RFC2119] 브래드너(S. Bradner), *요구사항수준을 나타내기 위하여*

RFC에서 사용하는 키워드 (Key words for use in RFCs to Indicate Requirement Levels), http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, 1997년3월.

[RFC 2396] 베르너스-리(T. Berners-Lee), 필딩(R. Fielding), 마신터(L. Masinter), *인터넷식별자(URI): 일반문법(Uniform Resource Identifiers (URI): Generic Syntax)*, http://www.ietf.org/rfc/rfc2396.txt, RFC 2396, 1988년8월.

[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, 2004년10월, http://www.w3.org/TR/xmlschema-1/

[XML Schema Part 2] XML Schema Part 2: Datatypes, W3C Recommendation, 2004년10월, http://www.w3.org/TR/xmlschema-2/

[XMLSpec] XML Specification, W3C Recommendation, 1998년2월, http://www.w3.org/TR/1998/REC-xml-19980210

6.4 비규범적 참조(Non-Normative References)

[BPEL 2.0] *Web Services Business Process Execution Language Version 2.0*. OASIS Standard. 2007년4월11일. 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

[XPATH 1.0] XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

[UNCEFACT XMLNDR] UN/CEFACT XML Naming and Design Rules Technical Specification, Version 3.0, http://www.unece.org/fileadmin/DAM/cefact/xml/UNCEFACT+XML+NDR+V3p0.pdf

6.5 조판 규칙

본 명세는 자원 데이터 모델을 서술하는 표에서 다음 규칙을 사용한다.

- 자원 명과 유형 명은(즉 “integer”, “string”과 같은 원자형 및 내장된 구조체의 이름) 이탤릭체로 표기한다.
- 속성명은 일반 폰트로 표기한다.

또한 본 명세는 다음 구문을 사용하여 자원을 직렬화한다.

- 이탤릭체로 표기된 값은 리터럴 값이 아니라 데이터형을 의미한다.
- 항목에 다음 글자를 추가하여 정규표현식과 유사한 방식으로 카디널리티를 표시한다.
  - "?" (0 또는 1)
  - "*" (0 이상)
  - "+" (1 이상)
- 수직 막대, "|"는 선택을 의미한다. 예를 들어, "a|b"는 "a"와 "b" 중의 선택을 의미한다.
- 괄호, "("와 ")"는 연산자 "?", "*", "+" 및 "|"의 범위를 표시한다.
- 생략 부호(즉 "...")는 확장 가능 여부를 표시한다. 생략 부호가 없다고 해서 확장할 수 없다는 것은 아니며 편의상 해당 부호를 생략한 것일 수 있다는 점에 주의한다.

6.6 네임스페이스

본 명세는 다수의 네임스페이스 접두사를 사용하며 이는 <표 6-1>에서 나열한 바와 같다. 네임스페이스 접두사는 임의적으로 선택한 것이며 의미론적으로 중요하지 않다는 것에 주의한다([XML] 참조). 뿐만 아니라, ‘tosca’ 네임스페이스(http://docs.oasis-open.org/tosca/ns/2011/12)를 기본 네임스페이스로 가정한다. 즉, 가독성을 위해 대응 네임스페이스 이름인 ‘tosca’를 본 명세에서 생략한다.

<표 6-1> 이 설명서에서 사용하는 접두사 및 네임스페이스

접두사	네임스페이스
tosca	http://docs.oasis-open.org/tosca/ns/2011/12
xs	http://www.w3.org/2001/XMLSchema

TOSCA가 정의하는 모든 정보 항목은 상기 XML 네임스페이스 URI[XML Namespaces] 중 하나로 확인할 수 있다. XML 네임스페이스 URI 중 하나를 역참조하여 TOSCA에 대한 규범적 XML 스키마([XML Schema Part 1][XML Schema Part 2]) 문서를 구할 수 있다.

6.7 확장성

TOSCA 확장성 메커니즘을 통하여 다음을 할 수 있다.

- 다른 네임스페이스의 속성은 어떠한 TOSCA 요소 상에도 표시할 수 있다.
- 다른 네임스페이스의 요소를 TOSCA 요소에 표시할 수 있다.
- 확장 속성과 확장 요소는 TOSCA 네임스페이스의 어떠한 속성 또는 요소의 의미와도 모순되지 않아야 한다(MUST NOT).

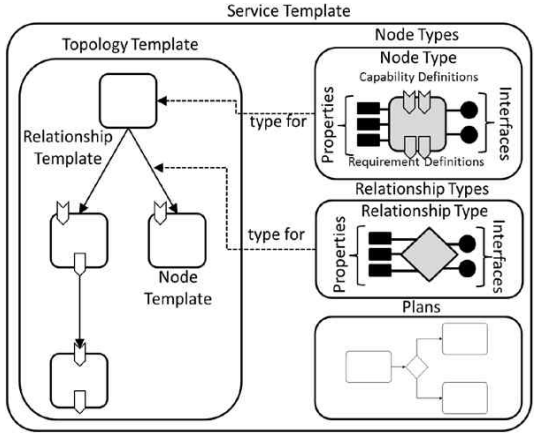
본 명세는 필수 확장과 선택 확장을 구별한다(다음 절은 확장을 선언하는 데 사용하는 구문에 대해 설명한다). 필수 확장을 사용하는 경우, 적합한 구현은 해당 확장을 이해하여야 한다(MUST). 선택 확장을 사용하는 경우, 적합한 구현은 해당 확장을 무시할 수도 있다(MAY).

7 핵심 개념 및 사용 패턴

본 절은 TOSCA의 주요 개념을 설명하고 서비스 템플릿의 몇몇 사용 패턴을 간략하게 제시한다.

7.1 핵심 개념

본 명세는 IT 서비스를 정의하기 위한 *메타모델(Metamodel)*을 정의한다. 메타모델은 서비스의 구조와 해당 서비스를 관리하는 방법을 정의한다. (서비스의 *토폴로지 모델*이라고도 불리는)토폴로지 템플릿은 서비스의 *구조(Structure)*를 정의한다. 계획은 서비스를 생성하고 종료하며 전체 생명주기 동안 서비스를 관리하는 데 사용하는 *처리 모델(Process Model)*을 정의한다. (그림 7-1)은 서비스를 정의하는 주요 요소에 대해 설명한다.



(그림 7-1) 서비스 템플릿의 구조적 요소와 그들 간의 관계

토폴로지 템플릿은 서비스의 토폴로지 모델을 (반드시 연결될 필요는 없는) 방향성 그래프(directed graph) 형태로 정의하는 *노드 템플릿(Node Template)*과 *관계 템플릿(Relationship Template)*으로 구성된다. 이러한 그래프의 노드는 노드 템플릿으로 표현하며 노드 템플릿은 *노드 유형(Node Type)*의 실체를 어느 서비스의 컴포넌트로 특정지은 것이다. 노드 유형은 노드 유형 *속성(Property)*을 통해 컴포넌트의 속성을 정의하고 *인터페이스(Interface)*를 통하여 해당 컴포넌트를 처리하기 위한 동작(Operation)을 정의한다. 재사용을 위해 노드 유형을 별도로 정의하도록 하며 특정 노드 템플릿은 특정 노드 유형을 가리키면서 사용 제약사항(예: 컴포넌트 생성 횟수)을 추가로 포함한다.

예를 들어, 애플리케이션 서버, 처리 엔진(process engine) 및 처리 모델(process model)로 구성된 어느 서비스를 고려한다. 토폴로지 템플릿은 “애플리케이션 서버”라는 노드 유형의 한 노드 템플릿, “처리 엔진”이라는 노드 유형의 다른 노드 템플릿 그리고 “프로세스 모델”이라는 노드 유형의 세 번째 노드 템플릿을 서비스에 포함한다고 정의한다. 애플리케이션 서버 노드 유형은 해당 유형의 인스턴스에 대한 IP 주소, 해당 IP 주소의 애플리케이션 서버를 설치하는 동작, 그리고 해당 서버 인스턴스를 정지하기 위한 동작 등의 속성들을 정의한다. 노드 템플릿의 제약 사항으로는 실제 애플리케이션 서버를 제공할 때 이용 가능한 IP 주소의 범위를 명시할 수 있다.

관계 템플릿은 토폴로지 템플릿에 있는 노드 간 관계의 실체를 명시한다. 각 관계 템플릿은 관계의 의미론(semantics)과 속성을 정의하는 *관계 유형(Relationship Type)*을 참조한다. 재사용을 위해 관계 유형을 별도로 정의한다. 관계 템플릿은 `SourceElement`와 `TargetElement` 요소 내에 출처(source)와 목표(target)를 정의함으로써 그것이 연결하는 요소들과 연결 방향을 표시한다. 또한 관계 템플릿은 선택적인(OPTIONAL) `RelationshipConstraints` 요소를 통해 제약 사항을 정의한다.

예를 들어 처리 엔진 노드 템플릿과 애플리케이션 서버 노드 템플릿 사이의 관계를 “호스트 된”이라는 의미로, 프로세스 모델 노드 템플릿과 처리 엔진 노드 템플릿 사이의 관계를 “배포된”이라는 의미로 확립할 수 있다.

배포된 서비스는 어떤 서비스 템플릿의 인스턴스다. 더 정확히 말하면 대체로 해당 서비스 템플릿에 대해 정의한 특별한 계획(흔히 빌드 계획이라고 지칭됨)을 실행함으로써 서비스 템플릿의 토폴로지 템플릿으로부터 해당 인스턴스를 만든다. 빌드 계획은 토폴로지 템플릿의 노드 템플릿과 관계 템플릿 내에 있는 여러 속성에 대한 실제 값을 제공한다. 이런 값은 사용자 인터페이스나 디렉터리 검색 등의 동작을 통해 사용자로부터 입력 받을 수 있으며 몇몇 속성에 대해서는 그 기본 값을 명시할 수도 있다. 빌드 계획은 일반적으로 노드 템플릿의 노드 유형에 정의된 동작을 활용한다.

예를 들어 설정한 IP 주소 범위 내에서의 실제 IP 주소를 이용하여 애플리케이션 서버를 설치함으로써 애플리케이션 노드 템플릿을 인스턴스화한다. 다음으로 (“호스트 된” 관계

템플릿이 나타내는 바와 같이) 애플리케이션 서버에 대한 실제 처리 엔진을 설치함으로써 처리 엔진 노드 템플릿을 인스턴스화한다. 마지막으로 (“배포된” 관계 템플릿이 나타내는 바와 같이) 상기 처리 엔진에 대한 처리 모델을 배포함으로써 처리 모델 노드 템플릿을 인스턴스화한다.

서비스 템플릿에서 정의한 계획은 서비스 인스턴스의 관리(특히 생성과 종료) 측면을 서술한다. 이런 계획을 처리 모델, 즉 하나 이상의 단계로 구성된 작업 흐름으로 정의한다. 본 명세는 처리 모델을 정의하기 위하여 다른 언어를 제공하기보다는 BPMN이나 BPEL과 같은 기존 표준 언어를 활용한다. 단, 기존 표준의 이용을 통해 이식성과 상호운용성을 촉진할 수 있지만 처리 모델 정의에 어떤 언어라도 사용할 수는 있다. TOSCA 메타 모델은 (계획 모델 참조를 통하여) 처리 모델을 참조하거나 (계획 모델을 통하여) 계획에 실제 모델을 포함하는 컨테이너(container)를 제공한다. 처리 모델은 노드 템플릿 인스턴스의 동작(또는 노드 템플릿의 `type` 속성에서 노드 템플릿이 각각 정의하는 동작), 관계 템플릿 인터페이스의 동작(또는 관계 템플릿의 `type` 속성에서 명시한 관계 유형이 각각 정의하는 동작) 또는 다른 인터페이스(가령 라이선스를 위한 외부 서버의 호출)를 참조하는 태스크를 포함할 수 있다. 이를 통하여 계획은 서비스 토폴로지의 노드를 직접 조작하거나 외부 시스템과 상호작용을 할 수 있다.

7.2 유스케이스

본 명세는 최소한 다음의 주요 유스케이스를 지원한다.

7.2.1 시장성 있는 실체로써의 서비스

서비스 템플릿의 표준화는 호스트 된 IT 서비스에 대한 시장 창출을 지원할 것이다. 특히 토폴로지 템플릿(서비스를 구성하는 일련의 컴포넌트와 그들 간의 의존성)을 명시하는 표준을 통하여 서비스 구조를 상호운용성 있게 정의할 수 있다. 특정 서비스의 내부 구조를 이해하고 있는 개발자는 이런 서비스 토폴로지 모델을 만들 수 있다. 잠재적인 고객이 선택하여 사용할 수 있도록 하나 이상의 서비스 제공자 카탈로그에 서비스 템플릿을 게시할 수 있다. 각 서비스 제공자는 실제 서비스 인스턴스를 지원하기 위하여 특정 서비스 토폴로지를 가용한 실제 인프라와 매핑하고 이에 따라 관리 계획을 조정할 것이다. 소위 인스턴스화 관리 계획 또는 빌드 계획이라고 알려진 해당 계획을 실행하여 실제 토폴로지 템플릿의 인스턴스를 생성할 수 있다. 서비스 템플릿을 만드는 서비스 개발자가 이런 빌드 계획을 제공할 수 있고 빌드 계획을 특정 서비스 제공자의 실제 환경에 맞춰 조정할 수 있다. 서비스의 전체 수명 주기의 다양한 상태에 대하여 유용한 다른 관리 계획을 서비스 템플릿의 일부로써 명시할 수 있다. 빌드 계획과 유사하게 이런 관리 계획도 특정 서비스 제공자의 실제 환경에 맞춰 조정할 수 있다.
따라서 서비스의 구조뿐만 아니라 관리 계획도 상호 운용이 가능한 방식으로 정의할 수 있다. 이런 계획은 특정 서비스에 대한 인스턴스의 생성 및 관리 방식을 서술한다. 서비스에 대하여 일련의 관리 계획을 정의함으로써 각 서비스를 관리하는 최선의 사례에 대해 재사용할 수 있는 지식을 제공하게 되어 서비스를 호스팅 하는 비용을 상당히 줄일 수 있다. 서비스를 모델링하는 사람이 분야에 대한 지식을 계획에 포함시킬 수 있는 반면에 이런 서비스를 사용하는 자는 이런 서비스를 “호출”함으로써 계획을 사용할 수 있다. 이를 통하여 기본 서비스 행동의 복잡성을 숨길 수 있고 이는 ITIL(IT Infrastructure Library) 명세를 만들게 된 상황과 매우 유사하다.

7.2.2 서비스 템플릿의 이식성

서비스 템플릿 표준화를 통해 IT 서비스를 정의하는 것에 대한 이식성을 지원한다. 여기서, 이식성은 다른 당사자(가령 클라우드 제공자, 기업 IT 부서 또는 서비스 개발자)가 만든 서비스 템플릿의 구조와 행동을 다른 클라우드 제공자가 이해하는 능력을 의미한다.

서비스의 이식성이 내부 컴포넌트의 이식성을 의미하지 않는다는 것에 주의한다. 서비스의 이식성은 상호 운용할 수 있는 방식으로 그 서비스의 정의를 이해할 수 있다는 것을 의미한다(즉, 표준을 준수하는 공급자는 토폴로지 모델과 이에 상응하는 계획을 이해한다). 특정 서비스를 구성하는 개별 컴포넌트 자체의 이식성은 다른 수단을 통하여 보장하여야 한다(개별 컴포넌트의 이식성이 서비스에 있어 중요한 경우).

7.2.3 서비스 구성

서비스 템플릿을 표준화하는 것은 비록 컴포넌트들을 다른 제공자(로컬 IT 부서 또는 다른 자동화 환경)가 호스트 하더라도 그 컴포넌트들로 서비스를 구성할 수 있도록 한다. 예를 들어 큰 조직은 다른 데이터 센터에 대하여 다른 공급자의 자동화 제품을 사용할 수 있다(가령 데이터 센터의 지리적 분산 또는 각 장소의 조직적 독립성 때문에). 서비스 템플릿은 호스팅 환경에 대한 어떠한 가정도 없이 추상성을 제공한다.

7.2.4 가상 이미지와의 관계

클라우드 제공자는 가상화한 미들웨어 스택에 기초하여 서비스를 호스트 할 수 있다. OVF [OVF] 패키지(package)와 같은 이미지 정의를 통하여 이런 미들웨어 스택을 표현할 수 있다. OVF를 사용하는 경우, OVF 패키지에서 정의하는 바와 같이 서비스 템플릿에 있는 노드는 가상 시스템이나 가상 시스템에서 실행중인 컴포넌트(OVF의 “제품”)에 대응된다. OVF 패키지가 복수의 가상 시스템을 포함하는 가상 시스템 집합을 정의하는 경우, 서비스 템플릿의 서브 트리는 OVF 가상 시스템 집합에 대응될 수 있다.

서비스 템플릿은 서비스 템플릿 요소와 OVF 패키지 요소 간의 결합(association)을 선언하는 방법을 제공한다. 이런 결합은 해당하는 OVF 패키지 요소를 배포하여 해당 서비스 템플릿 요소를 인스턴스화 할 수 있다는 것을 의미한다. 이런 결합은 OVF 패키지로 국

한하지 않으며 다른 패키지 유형이나 외부 서비스 인터페이스와도 가능하다. 이런 유연성은 다양한 가상화 기술, 서비스 인터페이스 및 사유 기술을 사용하여 서비스 템플릿을 구성할 수 있도록 한다.

7.3 서비스 템플릿 및 아티팩트

아티팩트(artifact)는 실행 파일(스크립트, 실행 프로그램, 이미지 등), 설정 파일 및 데이터 파일, 다른 실행 파일을 실행할 수 있기 위해서 필요한 것(라이브러리 등)과 같이 (서비스의) 배포를 실현하기 위해 필요한 콘텐츠를 의미한다. 아티팩트는 서로 다른 유형일 수 있으며(예: EJB 또는 python 스크립트) 그 콘텐츠는 유형에 따라 좌우된다. 전형적으로 아티팩트를 적절하게 처리하기 위하여 메타데이터(예: 적절한 실행 환경을 서술하는 것)를 함께 제공할 수도 있다.

TOSCA에서는 구현(implementation)과 배포(deployment) 두 가지 아티팩트로 구분한다. 구현 아티팩트는 어느 노드 유형의 동작에 대한 실행 파일을 의미하고 배포 아티팩트는 노드의 인스턴스를 구체화하기 위한 실행 파일을 의미한다. 예를 들어 이미지를 저장하는 REST 동작은 WAR 파일인 구현 아티팩트를 가질 수 있다. 이 REST 동작과 관련이 있는 노드 유형은 이미지 파일 자체를 배포 아티팩트로 가질 수 있다. 다음 두 가지 측면에서 구현 아티팩트와 배포 아티팩트는 근본적으로 차이가 있다.

1. 아티팩트를 배포하는 시점
2. 어떤 엔티티가 아티팩트를 어디에 배포할 것인지에 대한 사항

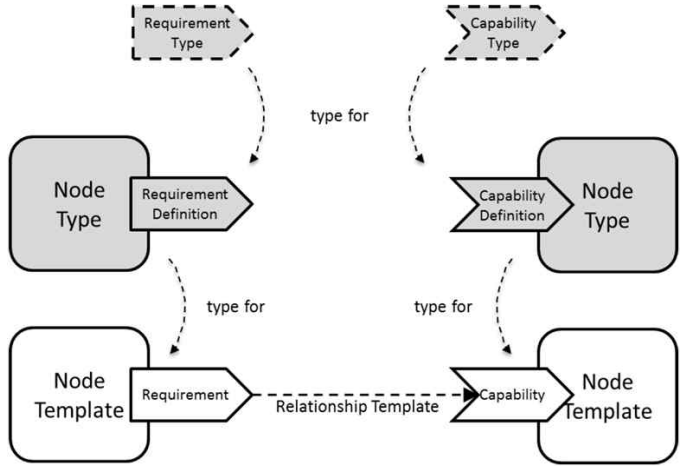
노드 유형의 동작은 해당 노드 유형(인스턴스)에 대한 관리 기능을 수행하고 이런 동작의 구현을 구현 아티팩트 형태로 제공할 수 있다. 따라서 관리 동작이 시작하기 전에 해당 동작의 구현 아티팩트를 관리 환경에 배포하여야 한다. 다시 말해서 "TOSCA 지원 환경"(소위 TOSCA 컨테이너)은 이런 관리 동작을 실행하는 데 필요한 일련의 구현 아티팩트 유형들을 모두 처리할 수 있어야 한다(MUST). 그러한 관리 동작 중 하나는 노드 유형을 인스턴스화 하는 것이다. 노드 유형을 인스턴스화하려면 목표 관리 환경에서 배포 아티팩트 제공이 필요한데 이를 위해 TOSCA 컨테이너는 처리 가능한 일련의 배포 아티팩트 유형들을 지원한다. 지원하지 않는 유형의 아티팩트(구현 또는 배포)를 포함하는 서비스 템플릿은 컨테이너가 처리할 수 없다(들여오기(import)할 때 오류가 발생).

7.4 요구사항 및 기능

TOSCA는 컴포넌트의 *요구사항*(requirement)과 *기능*(capability)을 표현할 수 있도록 한다. 예를 들어 어느 컴포넌트는 다른 컴포넌트가 제공하는 기능을 필요로 한다는 것을 표현하거나 어느 컴포넌트가 호스팅 환경에 대한 특정 요구사항(특정 자원의 할당이나 특정한 동작 모드의 지원여부 등)을 가지고 있다는 것을 표현하는 것이 가능하다.

특정한 유형의 *요구사항 정의(Requirement Definition)*와 *기능 정의(Capability Definition)*로 노드 유형을 표기하여 요구사항과 기능을 모델화한다. *요구사항 유형(Requirement Type)*과 *기능 유형(Capability Type)*을 재사용할 수 있는 실체로 정의하여 이런 정의를 여러 노드 유형에서 사용할 수 있게 된다. 예를 들어 데이터베이스 접속에 대한 클라이언트 요구사항을 서술하기 위하여 요구사항 유형 "DatabaseConnectionRequirement"를 정의할 수 있다. 그리고 이 요구사항 유형을 모든 종류의 노드 유형(예를 들어 데이터베이스에 접속하여야 하는 애플리케이션)을 위해 재사용할 수 있다.

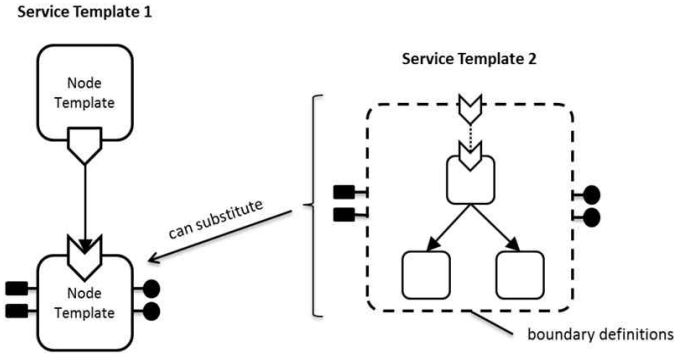
요구사항 정의나 기능 정의가 있는 노드 유형의 노드 템플릿은 해당 노드 템플릿이 고유한 콘텐츠로 각 요구사항과 기능을 표현한다. 예를 들어 요구사항 유형은 요구사항 메타데이터만 표현하는 반면에 노드 템플릿에서 표현하는 요구사항은 요구사항에서 정의하는 속성들에 대한 구체적인 값을 제공할 수 있다. 또한 어느 토폴로지 템플릿에 있는 노드 템플릿의 요구사항과 기능을 관계 템플릿으로 연결함으로써 한 노드의 특정 요구사항은 다른 노드가 제공하는 특정 기능으로 만족시킬 수 있다는 것을 표시할 수 있다. 이러한 모든 사항들은 (그림 7-2)와 같이 표현할 수 있고 요구사항은 다음과 같은 두 가지 방법으로 만족시킬 수 있다. (1) 어느 노드 템플릿의 요구사항은 동일 서비스 템플릿 내 다른 노드 템플릿의 기능으로 만족시킬 수 있는데 이때 관계 템플릿을 통해 요구사항-기능 쌍의 연결이 필요하다. (2) 어느 노드 템플릿의 요구사항은 일반 호스팅 환경(또는 TOSCA 컨테이너)으로 만족시킬 수 있다(예를 들어 인스턴스화하는 동안 노드 템플릿을 위해 필요한 자원을 할당한다).



(그림 7-2) 요구사항 및 기능

7.5 서비스 템플릿의 합성

7.4절에서 설명한 요구사항 및 기능의 개념에 의해 기존의 서비스 템플릿을 기반으로 새로운 서비스 템플릿을 구축하는 것이 가능하다. 예를 들어 애플리케이션 서버 계층에 호스트 된 비즈니스 애플리케이션에 대한 서비스 템플릿은 애플리케이션 자체의 구조와 행위를 정의하는 데 초점을 맞출 수 있다. 해당 애플리케이션을 호스트하는 애플리케이션 서버 계층의 구조는 애플리케이션 서버 배포 및 관리 전문의 다른 공급자가 구축한 별도의 서비스 템플릿으로 제공할 수 있다. 이러한 방식으로 공통 인프라스트럭처 템플릿을 재사용할 수 있다.



(그림 7-3) 서비스 템플릿의 합성

다른 서비스 템플릿을 사용하는 서비스 템플릿(즉, 상기 예제에서 비즈니스 애플리케이션 서비스 템플릿)의 관점에서 다른 서비스 템플릿(즉, 애플리케이션 서버 계층)은 마치 노드 템플릿처럼 보인다. 하지만 (그림 7-3)과 같이, 두 번째 서비스 템플릿이 노드 템플릿과 동일한 경계(즉, 속성, 기능 등)를 나타내는 경우, 두 번째 서비스 템플릿이 해당 노드 템플릿을 대체할 수 있다. 따라서 특정 노드 템플릿과 같은 *경계 정의(boundary definition)*만 가지고 있으면 어떠한 서비스 템플릿으로도 교체 가능하다는 것은 다양한 서비스 템플릿들을 유연하게 합성할 수 있다는 것을 뜻한다. 또한 이 개념을 통하여 서비스 템플릿 형태의 다른 대안을 제공할 수 있다. 예를 들어, 단일 노드 애플리케이션 서버 계층의 서비스 템플릿과 여러 개의 노드가 클러스터 된 애플리케이션 서버 계층의 서비스 템플릿이 존재할 수 있으며, 배포 시 적절한 옵션을 선택할 수 있다.

7.6 TOSCA의 정책

TOSCA의 정책을 통하여 비기능적 특성이나 서비스 품질(QoS)을 정의할 수 있다. 예를 들어 정책은 모니터링 특성, 지불 조건, 확장성 또는 가용성과 같은 다양한 사안을 표현할 수 있다.

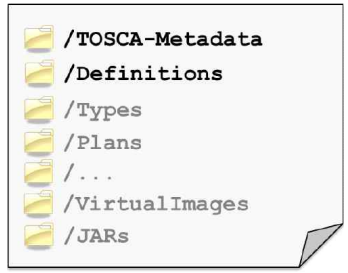
노드 템플릿을 각 인스턴스가 노출하는 비기능적 특성이나 서비스 품질을 표현하는 일련의 정책과 연결할 수 있다. 각 정책은 노드 템플릿의 개별 인스턴스에 대한 구체적인 지불 정보(지불 기간, 통화, 금액 등)와 같은 비기능적인 특성의 실제 속성 값을 명시한다. 정책 유형(Policy Type)을 통해 이러한 속성들을 정의할 수 있으며 특정 분야의 비기능적 특성이나 서비스 품질의 구조를 적절하게 반영하기 위하여 계층적으로 정의할 수 있다. 또한 하나의 정책 유형을 일련의 노드 유형들과 결합할 수 있다.

정책 템플릿(Policy Template)은 정책 유형에서 정의하는 유형의 실제 속성 값을 제공한다. 예를 들어 미국 클라이언트에 대한 월별 지불에 대한 정책 템플릿은 “지불 기간” 속성의 값을 “월별”로 그리고 “통화” 속성의 값을 “미국 달러(US$)”로 결정하고 “금액” 속성의 값은 결정하지 않는다. “금액” 속성의 값은 노드 템플릿의 정책을 결정하기 위하여 해당 정책 템플릿을 사용할 때 결정하게 된다. 따라서 가변 속성 값은 노드 템플릿에 있는 정책 템플릿을 실제로 사용할 때 결정되는 반면에 정책 템플릿은 정책의 불변 속성 값을 사전에 결정한다.

7.7 클라우드 애플리케이션을 위한 아카이브 포맷

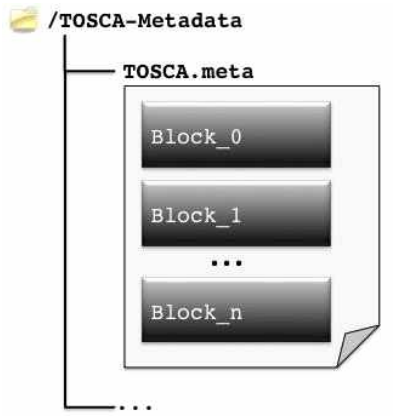
특정한 환경에서 클라우드 애플리케이션의 실행과 수명 주기의 관리를 지원하기 위해 해당 환경에서의 모든 해당 아티팩트를 사용할 수 있어야 한다. 이는 클라우드 애플리케이션의 서비스 템플릿뿐만 아니라 배포 아티팩트와 구현 아티팩트까지도 해당 환경에서 사용할 수 있어야 한다는 것을 의미한다. 이 모든 것의 사용 보장이 용이하도록 본 명세는 CSAR(Cloud Service ARchive, 클라우드 서비스 아카이브)라고 불리는 아카이브 포맷을 정의한다.

CSAR은 컨테이너 파일, 즉 유형이 다른 여러 파일들을 포함하며 일반적으로 이런 다양한 파일들은 여러 개의 하위 디렉터리로 구성하며 각 하위 디렉터리는 관련이 있는 파일(그리고 또 다른 하위 디렉터리 등)을 포함한다. 하위 디렉터리들로 구성된 디렉터리 구조와 모든 내부 콘텐츠는 특정 클라우드 애플리케이션에 특화된다. CSAR은 전형적으로 압축한 ZIP 파일이다.



(그림 7-4) CSAR의 구조

(그림 7-4)와 같이, 각 CSAR은 TOSCA-Metadata라는 이름의 하위 디렉터리를 포함하여야 하고(MUST) 이 하위 디렉터리는 *TOSCA 메타파일(meta file)*을 포함하여야 한다(MUST). 이 파일의 이름은 TOSCA이며 파일 확장자는 .meta인데 CSAR에 있는 다른 파일의 메타데이터를 나타내며 이름/값 쌍의 형식을 갖는다. 이런 이름/값의 쌍은 블록으로 구성되며 각 블록은 CSAR에 있는 특정 아티팩트의 메타데이터를 규정한다. 또한 빈 행을 통해 TOSCA 메타파일 내의 블록을 구분한다.



(그림 7-5) TOSCA 메타 파일의 구조

TOSCA 메타파일의 첫 번째 블록((그림 7-5)에 있는 Block_0)은 CSAR 자체의 메타데이터(버전, 작성자 등)를 규정한다. 나머지 블록의 경우는 첫 번째 이름/값의 쌍을 통해 아티팩트의 위치를 나타내고 나머지 이름/값의 쌍은 해당 아티팩트의 메타데이터(예: MIME 유형의 아티팩트)를 나타낸다((그림 7-6) 참고).

(그림 7-6) 아티팩트에 대한 메타데이터의 규정

8 TOSCA 정의서

TOSCA 서비스 템플릿 자체뿐만 아니라 서비스 템플릿을 정의하는 데 필요한 모든 요소(노드 유형 정의, 관계 유형 정의 등)는 TOSCA 정의서에서 제공한다. 이 절은 TOSCA 정의서의 전체적인 구조, 확장 메커니즘 및 들여오기 기능을 설명한다. 이후 절에서는 서비스 템플릿, 노드 유형, 노드 유형 구현, 관계 유형, 관계 유형 구현, 요구사항 유형, 기능 유형, 아티팩트 유형, 아티팩트 템플릿, 정책 유형 및 정책 템플릿에 대하여 자세히 설명한다.

8.1 XML 구문

다음 의사 스키마는 정의서의 XML 구문을 정의한다.

```
<Definitions id="xs:ID"
             name="xs:string"?
             targetNamespace="xs:anyURI">

  <Extensions>
    <Extension namespace="xs:anyURI"
               mustUnderstand="yes|no"?/> +
  </Extensions> ?

  <Import namespace="xs:anyURI"?
          location="xs:anyURI"?
          importType="xs:anyURI"/> *

  <Types>
    <xs:schema .../> *
  </Types> ?

  (
    <ServiceTemplate> ... </ServiceTemplate>
  |
    <NodeType> ... </NodeType>
  |
    <NodeTypeImplementation> ... </NodeTypeImplementation>
  |
    <RelationshipType> ... </RelationshipType>
  |
    <RelationshipTypeImplementation> ... </RelationshipTypeImplementation>
  |
    <RequirementType> ... </RequirementType>
  |
    <CapabilityType> ... </CapabilityType>
  |
    <ArtifactType> ... </ArtifactType>
  |
    <ArtifactTemplate> ... </ArtifactTemplate>
  |
    <PolicyType> ... </PolicyType>
  |
    <PolicyTemplate> ... </PolicyTemplate>
  ) +

</Definitions>
```

8.2 속성

Definitions 요소는 다음 속성을 가진다.

- id: 이 속성은 해당 정의서의 식별자를 명시하는데 이러한 식별자는 목표 네임스페이스 내에서 고유해야 한다(MUST).

- name: 이 선택적(OPTIONAL) 속성은 해당 정의서의 서술적인 이름을 명시한다.

- targetNamespace: 이 속성의 값은 해당 정의서에 대한 목표 네임스페이스를 명시한다. 자체 targetNamespace 속성으로 본 속성을 재 정의한 경우를 제외하고는 해당 정의서 내에 정의한 모든 요소를 이 네임스페이스에 추가한다.

- Extensions: 이 선택적(OPTIONAL) 요소는 TOSCA 확장 속성과 확장 요소의 네임스페이스를 명시한다. Extensions 요소는 최소한 하나 이상의 Extension 요소를 포함하여야 하며(MUST) Extension 요소는 다음 속성을 가진다.

  - namespace: 이 속성은 TOSCA 확장 속성과 확장 요소의 네임스페이스를 명시한다.

  - mustUnderstand 이 선택적(OPTIONAL) 속성의 값 본 확장이 반드시 구현에서 이해되어야 하는지를 명시한다. 이 속성의 값이 “yes”(이 속성에 대한 기본 값)인 경우, 확장은 의무적이고 그렇지 않으면 확장은 선택 사항이다. TOSCA 구현이 하나 이상의 의무 확장을 지원하지 아니하는 경우, 해당 정의서를 거부하여야 한다(MUST). 선택적인 확장을 무시할 수 있으며(MAY) 선택적인 확장을 반드시 선언할 필요는 없다. Extensions 요소에서 동일한 확장 URI를 여러 번 선언할 수 있다. 하나의 Extension 요소에서 확장 URI를 의무 확장으로 확인하고 다른 확장 URI에서 선택적인 것으로 확인하는 경우, 의무적인 의미가 우선하며 이를 실시하여야 한다(MUST). 어느 Extensions 요소에서 확장을 선언하는 것을 순서가 없는 것으로 취급하여야 한다(MUST).

- Import: 이 요소는 외부 TOSCA 정의, XML 스키마 정의 또는 WSDL 정의에 대한 종속성을 선언한다. 어떤 수의 Import 요소도 Definitions 요소의 자식으로 나타날 수 있으며(MAY) Import 요소는 다음과 같은 속성을 가진다.

  - namespace: 이 선택적(OPTIONAL) 속성은 들여온 정의를 확인하는 절대 URI를 명시한다. namespace 속성이 없는 Import 요소는 외부 정의(네임스페이스에 고유하지 아니한)를 사용하고 있다는 것을 의미한다. namespace 속성을 명시하는 경우, 불러온 정의가 해당 네임스페이스에 있어야 한다(MUST). namespace를 명시하지 아니한 경우, 불러온 정의는 targetNamespace를 포함하지 아니하여야 한다(MUST NOT). 네임스페이스 http://www.w3.org/2001/XMLSchema는 암묵적으로 들여오지만 http://www.w3.org/2001/XMLSchema에 대하여 암묵적인 XML 네임스페이스 접두사를 정의하지 않는다는 것에 주의한다.
  - location: 이 선택적(OPTIONAL) 속성은 관련 정의를 포함하는 문서의 위치를 의미하는 URI를 포함한다. 이런 위치 URI 기본의 일반 규칙[XML Base, RFC 2396]에 따라 상대 URI일 수 있다(MAY). location 속성이 없는 Import 요소는 외부 정의를 사용하되 해당 외부 정의의 위치에 대하여 진술하지 않는다는 것을 의미한다. location 속성은 일종의 힌트이며 적합한 TOSCA 구현이 특정한 위치에서 들여온 문서를 조회할 의무는 없다.

  - importType: 이 필수(REQUIRED) 속성은 문서에서 사용되는 인코딩 언어를 식별하는 절대 URI를 제공함으로써 들여온 문서의 종류를 확인한다. 서비스 템플릿 문서를 불러올 때 importType 속성의 값을 http://docs.oasis-open.org/tosca/ns/2011/12로 정하여야 하고 WSDL 1.1 문서를 불러올 때는 http://schemas.xmlsoap.org/wsdl/로 하여야 하며 XSD 문서를 불러올 때는 http://www.w3.org/2001/XMLSchema로 하여야 한다(MUST).

이런 규칙에 따르면 namespace 및 location 속성 없이 importType 속성만 갖는 Import 요소가 있을 수 있으며 그러한 Import 요소는 네임스페이스가 보증되지 않으면서 정의된 곳을 특정할 수 없는 외부 정의를 사용하고 있다는 것을 의미한다.

정의서는 사용하고 있는 모든 노드 유형, 노드 유형 구현, 관계 유형, 관계 유형 구현, 요구사항 유형, 기능 유형, 아티팩트 유형, 정책 유형, WSDL 정의 및 XML 스키마 문서를 정의하거나 들여와야 한다(MUST). 여러 문서에서 네임스페이스를 정의한 것을 사용하는 것을 지원하기 위하여, 정의서는 동일한 namespace와 importType에 대하여 하나 이상의 들여오기 선언을 포함할 수 있다(MAY). 정의서가 특정 namespace와 importType에 대하여 하나 이상의 들여오기를 선언하는 경우, 각 선언은 서로 다른 location 값을 포함하여야 한다(MUST). Import 요소는 개념상으로 순서가 없다. 들여온 정의서가 들여오는 정의서가 사용하는 컴포넌트와 상충하는 정의를 포함하는 경우, 해당 정의서를 거부하여야 한다(MUST).

TOSCA를 준수하는 구현은 기존의 들여온 문서(또는 네임스페이스)가 들여오는 문서(또는 네임스페이스)를 전이적으로 불러오지는 않는다. 특히 이는 정의서에 있는 요소가 외부 항목을 사용하는 경우, 해당 항목을 정의하는 문서(또는 네임스페이스)가 정의서를 직접적으로 불러와야 한다는 것을 의미한다(MUST). 이 요구사항은 불러온 문서 다른 문서나 네임스페이스를 불러올 수 있는 능력 자체를 제한하는 것은 아니다.

- Types: 이 요소는 정의서에서 도입한 XML 정의를 명시한다. 하나 이상의 별도의 스키마 정의(일반적으로 xs:schema 요소)에서 이런 정의를 규정한다. Types 요소는 별도의 파일에서 XML을 정의하고 이를 들여올 필요 없이 정의서 내에서 XML을

정의한다. Types 요소에 내포된 xs:schema 요소는 유효한 XML 스키마 정의이어야 한다(MUST). 내포된 xs:schema 요소의 targetNamespace 속성을 명시하지 아니하는 경우, 이 요소 내 모든 정의는 Definitions 요소를 포함하는 목표 네임스페이스의 일부가 된다. 단, 본 명세는 Types 요소에 내포된 모든 유형 시스템을 사용하는 것을 지원하지만 TOSCA를 준수하는 구현은 xs:schema에 대한 지원만 요한다(REQUIRED).

비고: 설명서는 Types 요소에 내포된 모든 유형 시스템을 사용하는 것을 지원한다. 하지만 TOSCA를 준수하는 구현은 xs:schema를 지원하여야 한다.

- ServiceTemplate: 이 요소는 어느 클라우드 애플리케이션에 대한 완전한 서비스 템플릿를 명시한다. 서비스 템플릿은 클라우드 애플리케이션의 토폴로지 템플릿과 몇몇 계획에 대한 정의를 포함한다. 해당 서비스 템플릿 내에서, 동일 (들여온) 정의서의 어떠한 유형 정의(노드 유형, 관계 유형 등)도 사용할 수 있다.

- NodeType: 이 요소는 어느 서비스 템플릿의 노드 템플릿을 위해 참조될 수 있는 노드의 유형을 명시한다.

- NodeTypeImplementation: 이 요소는 어느 서비스 템플릿의 노드 템플릿을 위해 참조될 수 있는 노드의 유형의 관리 기능에 대한 구현을 명시한다.

- RelationshipType: 이 요소는 어느 서비스 템플릿의 관계 템플릿을 위해 참조될 수 있는 관계의 유형을 명시한다.

- RelationshipTypeImplementation: 이 요소는 어느 서비스 템플릿의 관계 템플릿을 위해 참조될 수 있는 관계의 유형의 관리 기능에 대한 구현을 명시한다.

- RequirementType: 이 요소는 어느 서비스 템플릿에서 사용되는 노드 유형이 노출할 수 있는 요구사항의 유형을 명시한다.

- CapabilityType: 이 요소는 어느 서비스 템플릿에서 사용되는 노드 유형이 노출할 수 있는 기능의 유형을 명시한다.

- ArtifactType: 이 요소는 서비스 템플릿에서 사용되는 아티팩트의 유형을 명시한다. 예를 들어 아티팩트 유형은 WAR 파일(확장자: war)이나 EAR 파일(확장자: ear)과 같은 애플리케이션 모듈, RPM과 같은 OS 패키지 또는 OVA 파일(확장자: ova)과 같은 가상 머신 이미지일 수 있다.

- ArtifactTemplate: 이 요소는 서비스 템플릿의 일부가 참조하는 아티팩트를 서술

하는 템플릿을 명시한다. 예를 들어 애플리케이션 서버 노드를 위해 설치할 수 있는 아티팩트를 아티팩트 템플릿으로써 정의할 수 있다.

- PolicyType: 이 요소는 어느 서비스 템플릿에서 정의한 노드 템플릿들과 결합될 수 있는 정책의 유형을 명시한다. 예를 들어 웹 서버 계층에 있는 노드들의 확장 정책을 정책 유형으로써 정의할 수 있고 이러한 유형에 확장 정책이 가질 수 있는 속성들을 명시한다.

- PolicyTemplate: 이 요소는 어느 서비스 템플릿에서 정의한 노드 템플릿과 연결될 수 있는 정책의 템플릿을 명시한다. 정책 유형과 달리, 정책 템플릿은 정책 템플릿이 참조하는 정책 유형이 명시하는 일련의 속성에 따라 하나의 정책에 대한 명확한 값을 정의할 수 있다.

TOSCA 정의서는 ServiceTemplate, NodeType, NodeTypeImplementation, RelationshipType, RelationshipTypeImplementation, RequirementType, CapabilityType, ArtifactType, ArtifactTemplate, PolicyType 또는 PolicyTemplate 중 하나 이상의 요소를 정의해야 하고(MUST), 하나 이상의 요소를 임의의 순서로 정의할 수도 있다.

이러한 기법은 서비스 템플릿을 모듈 단위로 정의할 수 있게 한다. 예를 들어 하나의 정의서는 노드 유형 및 관계 유형 정의만 포함할 수 있는데 이러한 정의서는 해당 노드 유형 및 관계 유형을 사용하여 서비스 템플릿을 정의하는 다른 정의서로 들여올 수도 있다. 마찬가지로 노드 유형 속성들을 별도의 XML 스키마 정의서 상에 정의하고 노드 유형을 정의할 때 이를 들여오거나 참조할 수도 있다.

모든 TOSCA 요소는 documentation 요소를 사용하여 사용자에게 주석을 제공할 수 있다(MAY). 그 내용은 평문, HTML 등으로 구성될 수 있다. documentation 요소는 선택(OPTIONAL) 사항이며 다음 구문을 따른다.

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

documentation 요소의 사용 예시는 다음과 같다.

```
<Definitions id="MyDefinitions" name="My Definitions" ...>

  <documentation xml:lang="EN">
    This is a simple example of the usage of the documentation
    element nested under a Definitions element. It could be used,
    for example, to describe the purpose of the Definitions document
    or to give an overview of elements contained within the Definitions
    document.
  </documentation>

</Definitions>
```

8.3 예시

다음 정의서는 두 노드 유형인 “Application”과 “ApplicationServer” 그리고 한 관계 유형인 “ApplicationHostedOnApplicationServer”를 정의한다. 두 노드 유형에 대한 속성 정의는 별도의 XML 스키마 정의서 파일에 명시되어 있으며 이 파일은 **Import** 요소를 통하여 해당 정의서로 들여온다.

```
<Definitions id="MyDefinitions" name="My Definitions"
  targetNamespace="http://www.example.com/MyDefinitions"
  xmlns:my="http://www.example.com/MyDefinitions">

  <Import importType="http://www.w3.org/2001/XMLSchema"
    namespace="http://www.example.com/MyDefinitions">

  <NodeType name="Application">
    <PropertiesDefinition element="my:ApplicationProperties"/>
  </NodeType>

  <NodeType name="ApplicationServer">
    <PropertiesDefinition element="my:ApplicationServerProperties"/>
  </NodeType>

  <RelationshipType name="ApplicationHostedOnApplicationServer">
    <ValidSource typeRef="my:Application"/>
    <ValidTarget typeRef="my:ApplicationServer"/>
  </RelationshipTemplate>

</Definitions>
```

9 서비스 템플릿

이 절은 *서비스템플릿*을 정의하는 방법을 명시한다. 서비스템플릿은 토폴로지템플릿을 통하여 클라우드 애플리케이션의 구조를 서술하고 클라우드 애플리케이션의 관리 가능한 행동을 계획 형태로 정의한다.

토폴로지 템플릿에서 정의하는 노드템플릿과 같은 서비스템플릿에 있는 요소는 서비스템플릿을 포함하는 형태의 정의문서에서 정의할 수 있거나, 별도로 불러온 정의문서에서 정의할 수 있는 노드유형 형태로 TOSCA 요소를 참조한다.

클라우드 애플리케이션을 배포하고 관리하는데 직접 사용하기 위하여 서비스템플릿을 정의할 수 있다. 또는 더 큰 서비스 템플릿을 구성하는데 서비스 템플릿을 사용 할 수 있다.

9.1 XML 구문

다음 의사 스키마는 서비스 템플릿의 XML 문법을 정의한다.


```
<ServiceTemplate id="xs:ID"
                 name="xs:string"?
                 targetNamespace="xs:anyURI"
                 substitutableNodeType="xs:QName"?>

 <Tags>
   <Tag name="xs:string" value="xs:string"/> +
 </Tags> ?

  <BoundaryDefinitions>
    <Properties>
      XML fragment
      <PropertyMappings>
        <PropertyMapping serviceTemplatePropertyRef="xs:string"
                         targetObjectRef="xs:IDREF"
                         targetPropertyRef="xs:string"/> +
      </PropertyMappings/> ?
    </Properties> ?

    <PropertyConstraints>
      <PropertyConstraint property="xs:string"
                          constraintType="xs:anyURI"> +
        constraint ?
      </PropertyConstraint>
    </PropertyConstraints> ?

    <Requirements>
      <Requirement name="xs:string"? ref="xs:IDREF"/> +
    </Requirements> ?

    <Capabilities>
      <Capability name="xs:string"? ref="xs:IDREF"/> +
    </Capabilities> ?

    <Policies>
      <Policy name="xs:string"? policyType="xs:QName"
              policyRef="xs:QName"?>
        policy specific content ?
      </Policy> +
    </Policies> ?

    <Interfaces>
      <Interface name="xs:NCName">
        <Operation name="xs:NCName">
          (
            <NodeOperation nodeRef="xs:IDREF"
                           interfaceName="xs:anyURI"
                           operationName="xs:NCName"/>
          |
            <RelationshipOperation relationshipRef="xs:IDREF"
                                   interfaceName="xs:anyURI"
                                   operationName="xs:NCName"/>
          |
            <Plan planRef="xs:IDREF"/>
          )
        </Operation> +
      </Interface> +
```

```
  </Interfaces> ?

</BoundaryDefinitions> ?

<TopologyTemplate>
  (
   <NodeTemplateid="xs:ID" name="xs:string"? type="xs:QName"
               minInstances="xs:integer"?
               maxInstances="xs:integer| xs:string"?>
     <Properties>
      XML fragment
     </Properties> ?

     <PropertyConstraints>
      <PropertyConstraint property="xs:string"
                       constraintType="xs:anyURI">
       constraint ?
      </PropertyConstraint> +
     </PropertyConstraints> ?

     <Requirements>
      <Requirement id="xs:ID" name="xs:string" type="xs:QName"> +
       <Properties>
        XML fragment
       <Properties> ?
       <PropertyConstraints>
        <PropertyConstraint property="xs:string"
                          constraintType="xs:anyURI"> +
         constraint ?
        </PropertyConstraint>
       </PropertyConstraints> ?
      </Requirement>
     </Requirements> ?

     <Capabilities>
      <Capability id="xs:ID" name="xs:string" type="xs:QName"> +
       <Properties>
        XML fragment
       <Properties> ?
       <PropertyConstraints>
        <PropertyConstraint property="xs:string"
                          constraintType="xs:anyURI">
          constraint ?
         </PropertyConstraint> +
        </PropertyConstraints> ?
      </Capability>
     </Capabilities> ?

     <Policies>
      <Policy name="xs:string"? policyType="xs:QName"
            policyRef="xs:QName"?>
       policy specific content ?
      </Policy> +
     </Policies> ?

     <DeploymentArtifacts>
      <DeploymentArtifact name="xs:string" artifactType="xs:QName"
```



```
                          artifactRef="xs:QName"?>
          artifact specific content ?
       </DeploymentArtifact> +
     </DeploymentArtifacts> ?
    </NodeTemplate>
   |
    <RelationshipTemplate id="xs:ID" name="xs:string"?
                      type="xs:QName">
     <Properties>
      XML fragment
     </Properties> ?

     <PropertyConstraints>
      <PropertyConstraint property="xs:string"
                       constraintType="xs:anyURI">
        constraint ?
       </PropertyConstraint> +
     </PropertyConstraints> ?

     <SourceElement ref="xs:IDREF"/>
     <TargetElement ref="xs:IDREF"/>

     <RelationshipConstraints>
      <RelationshipConstraint constraintType="xs:anyURI">
       constraint ?
      </RelationshipConstraint> +
     </RelationshipConstraints> ?

    </RelationshipTemplate>
  ) +
</TopologyTemplate>

<Plans>
  <Plan id="xs:ID"
       name="xs:string"?
       planType="xs:anyURI"
       planLanguage="xs:anyURI">

    <Precondition expressionLanguage="xs:anyURI">
     condition
    </Precondition> ?

    <InputParameters>
     <InputParameter name="xs:string" type="xs:string"
                  required="yes|no"?/> +
    </InputParameters> ?

    <OutputParameters>
     <OutputParameter name="xs:string" type="xs:string"
                   required="yes|no"?/> +
    </OutputParameters> ?

    (
     <PlanModel>
      actual plan
     </PlanModel>
     |
```

```
172         <PlanModelReference reference="xs:anyURI"/>
173       )
174
175     </Plan> +
176   </Plans> ?
177
178 </ServiceTemplate>
```

9.2 속성

ServiceTemplate 요소는 다음과 같은 특성을 가진다.

- id: 이 속성은 서비스 템플릿의 식별자를 명시한다. 서비스 템플릿의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- name: 이 선택적인(OPTIONAL) 속성은 서비스 템플릿의 서술적인 이름을 명시한다.

- targetNamespace: 이 선택적인(OPTIONAL) 속성의 값은 서비스 템플릿에 대한 목표 네임스페이스를 명시한다. 이를 명시하지 아니 하는 경우, Definitions 요소의 targetNamespace 요소를 서비스 템플릿을 선언한 네임스페이스에 추가한다.

- substitutableNodeType: 이 선택적인(OPTIONAL) 속성은 이 서비스 템플릿이 대체할 수 있는 노드유형을 명시한다. 다른 서비스 템플릿이 명시한 노드유형(또는 이 노드유형이 파생된 다른 노드 유형)의 노드 템플릿을 포함하는 경우, 대체된 노드의 기능을 규정하는 이 서비스 템플릿의 인스턴스가 이 노드 템플릿을 대체할 수 있다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(다른 요소가 서비스 템플릿을 서술하는데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다.
  Tag요소는 다음 특성을 가진다.
  – name: 이 속성은 태그의 이름을 명시한다.
  – value: 이 속성은 태그의 값을 명시한다.

  비고: 이 태그에서 정의하는 이름/값의 쌍은 규범적인 해석이 아니다.

- BoundaryDefinitions: 이 선택적인(OPTIONAL) 요소는 서비스 템플릿이 그 경계를 초과하여 노출하는 특성(즉 서비스 템플릿의 밖에서도 알 수 있는 특성)을 명시한다. BoundaryDefinitions 요소는 다음 특성을 가진다.

  ○ Properties: 이 선택적인(OPTIONAL) 요소는 서비스 템플릿의 전역특성을

Properties 요소의 본문에 포함된 XML 프래그먼트 형태로 명시한다. 이런 특성을 서비스 템플릿에 있는 컴포넌트의 특성으로 매핑하여 이를 외부에 노출할 수 있다. Properties 요소는 다음 특성을 가진다.

  ▪ PropertyMappings: 이 선택적인(OPTIONAL) 요소는 서비스 템플릿의 특성 중 하나 이상을 서비스 템플릿(가령 노드 템플릿, 관계 템플릿 등)에 있는 컴포넌트의 특성으로 매핑하는 것을 명시한다. 별도의 내포된 PropertyMapping 요소를 통하여 각 특성 매핑을 정의한다. PropertyMapping 요소는 다음 특성을 가진다.

    - serviceTemplatePropertyRef: 이 속성은 서비스 템플릿의 특성을 정의하는 XML 프래그먼트에서 평가 할 XPath 표현식을 통하여 서비스 템플릿의 특성을 확인한다.

    - targetObjectRef: 이 속성은 각 서비스 템플릿의 속성이 매핑하는 특성을 규정하는 객체를 명시한다. 참조한 목표 객체는 노드 템플릿, 노드 템플릿의 요구사항, 노드 템플릿의 기능 또는 관계 템플릿 중 하나이어야 한다.

    - targetPropertyRef: 이 속성은 목표 객체의 특성을 정의하는 XML 프래그먼트에서 평가할 XPath 표현식을 통하여 목표 객체의 특성을 확인한다.

      비고: 서비스 템플릿 특성을 서비스 템플릿에 있는 컴포넌트의 특성으로 매핑하는 경우, 서비스 템플릿 특성의 XML 스키마 유형과 매핑 한 특성은 서로 호환이 되어야 한다.

      비고: 서비스 템플릿 특성을 서비스 템플릿에 있는 컴포넌트의 특성으로 매핑하는 경우, 서비스 템플릿 특성을 읽는 것은 매핑 한 특성을 읽는 것에 따르고 서비스 템플릿 특성을 쓰는 것은 매핑 한 특성을 쓰는 것에 따른다.

○ PropertyConstraints: 이 선택적인(OPTIONAL) 요소는 하나이상의 서비스 템플릿 특성에 대한 제약사항을 명시한다. 별도의 내포된 PropertyConstraint요인을 통하여 각 제약사항을 명시한다. PropertyConstraint 요인은 다음특성을 가진다.

  ▪ property: 이 속성은 서비스 템플릿의 특성을 정의하는 XML 프래그먼트에서 평가할 XPath 표현식을 통하여 특성을 확인한다.

비고: 특성 제약 사항에 의하여 영향을 받는 특성을 서비스 템플릿에 있는 컴포넌트의 특성으로 매핑하는 경우, 특성 제약 사항은 매핑된 특성에 대하여 정의한 특성 제약 사항과 호환이 되어야 한다.

- constraintType: 이 속성은 URI를 (제약사항의 의미와 콘텐츠의 포맷을 정의하는) 통하여 제약사항의 종류를 명시한다.

- PropertyConstraint 요소의 본문은 실제 제약사항을 규정한다.
  비고: constraintType URI가 제약 사항을 이미 적절하게 명시한 경우, 본문을 비워 둘 수 있다. 예를 들어 “읽기 전용” 제약 사항은 constraintType URI를 통해서만 표현할 수 있다

○ Requirements: 이 선택적인(OPTIONAL) 요소는 서비스 템플릿이 노출하는 요구사항을 명시한다. 이런 요구사항은 서비스 템플릿의 경계를 넘어 전파되는 서비스 템플릿에 있는 노드 템플릿의 요구사항과 일치한다. 별도의 내포된 Requirements 요소를 통하여 각 요구사항을 정의한다. Requirement 요소는 다음 특성을 가진다.

- name: 이 선택적(OPTIONAL) 속성을 통하여 노드 템플릿 중 참조한 Requirement가 명시하는 것을 이외의 요구사항이름을 명시할 수 있다.

- ref: 이 속성은 서비스 템플릿에 있는 노드 템플릿의 Requirement요소를 참조한다.

○ Capabilities: 이 선택적인(OPTIONAL) 요소는 서비스 템플릿이 노출하는 기능을 명시한다. 이런 기능은 서비스 템플릿의 경계를 넘어 전파되는 서비스 템플릿에 있는 노드 템플릿의 기능과 일치한다. 별도의 내포된 Capabilities 요소를 통하여 각 기능을 정의한다. Capability 요소는 다음 특성을 가진다.

- name: 이 선택적(OPTIONAL) 속성을 통하여 노드 템플릿 중 참조한Capability가 명시 하는 것을 이 외의 기능 이름을 명시할 수 있다.

- ref: 이 속성은 서비스 템플릿에 있는 노드 템플릿의 Capability 요소를 참조한다.

○ Policies: 이 선택적(OPTIONAL) 요소는 특정 관리측면과 관련한 서비스 템플릿의 전역 정책을 명시한다. Policies TOSCA 구현을 통하여 실시

하여야 한다(즉 정책을 AND로 결합한다). 별도의 내포된 Policy 요소를 통하여 각 정책을 정의한다. Policy요소는 다음 특성을 가진다.

- name: 이 선택적(OPTIONAL) 속성을 통하여 정책을 정의할 수 있다. 이 이름은 Policies 요소에서 고유하여야 한다.

- policyType: 이 속성은 이 정책의 유형을 명시한다. 이 속성의 QName 값은 같은 정의 문서나 불러온 문서에서 정의한 PolicyType의 QName과 일치하여야 한다(SHOULD). policyType 속성은 Policy 요소에 특정한 아티팩트 유형을 명시하고 정책이 policyRef 속성을 통하여 참조하는 정책 템플릿의 유형을 보여준다.

- policyRef: 이 선택적인(OPTIONAL) 속성의 QName 값은 서비스 템플릿과 관련이 있는 정책 템플릿을 참조한다. 같은 TOSCA 정의문서나 별도의 문서에서(현재 정의문서로 불러온) 이 정책 템플릿을 정의할 수 있다. policyRef 속성이 참조하는 정책 템플릿의 유형은 policyType 속성에서 명시한 유형과 같은 유형이거나 그 하위 유형이어야 한다.

  비고: 정책 템플릿을 참조하지 아니하는 경우, Policy 요소의 특정 콘텐츠가 서비스 템플릿의 맥락에서 충분한 정책 정보를 표현한다고 추정한다.

  비고: 정책 템플릿이 비기능적인 행동에 대한 불변 정보(즉 사용 가능성 정책의 사용 가능성 클래스와 같이 맥락과 상관이 없는 정보)를 제공하는 반면에 서비스 템플릿에서 정의하는 Policy 요소는 Policy 요소의 특정 정책 본문에서 가변 정보(즉 서비스의 사용 가능성을 점검하기 위한 특정한 프레임워크 주파수와 같이 맥락에 고유한 정보)를 제공할 수 있다.

○ Interfaces: 이 선택적(OPTIONAL) 요소는 서비스 템플릿에서 만들어진 서비스 인스턴스에서 호출할 수 있는 동작이 있는 인터페이스를 명시한다. Interfaces 요소는 다음 특성을 가진다.

- Interface: 이 요소는 서비스 템플릿이 노출하는 하나의 인터페이스를 명시한다. Interface 요소는 다음 특성을 가진다.

- name: 이 속성은 서비스 템플릿의 경계 정의의 범위에서 고유하여야하는 URI나 NCName으로써 인터페이스의 이름을 명시한다(MUST).

- Operation: 이 요소는 서비스 템플릿이 노출하는 인터페이스의

동작하나를 명시한다. 서비스 템플릿이 노출하는 동작을 서비스 템플릿의 내부 컴포넌트(실제로 동작하는)로 매핑한다. 이를 노드템플릿이 규정하는 동작(즉 노드 템플릿의 type속성에서 명시하는 노드유형이 정의하는 동작), 관계 템플릿이 규정하는 동작(즉 관계 템플릿의 type속성에서 명시하는 관계유형이 정의하는 동작) 또는 서비스 템플릿의 계획으로 매핑할 수 있다. 노출된 동작을 서비스 템플릿에서 만드는 서비스 인스턴스에서 호출하는 경우, 노출된 동작에 매핑된 동작이나 계획을 실제로 호출한다. Operation 요소는 다음 특성을 가진다.

- name: 이 속성은 동작의 이름을 명시한다. 이 이름은 인터페이스에서 고유하여야 한다(MUST).

- NodeOperation: 이 요소는 노드 템플릿의 동작에 대한 참조를 명시한다. 이 요소의 nodeRef 속성은 각 노드 템플릿에 대한 참조를 명시한다. 서비스 템플릿이 노출하는 동작에 매핑된 특정한 인터페이스와 동작을 각각 interfaceName 속성과 operationName 속성을 통하여 명시한다.

  비고: 참조한 노드 템플릿의 type 속성에서 정의한 노드 유형은 인터페이스와 특정한 이름의 동작을 정의하여야 한다.

- RelationshipOperation: 이 요소는 관계 템플릿의 동작에 대한 참조를 명시한다. 이 요소의 relationshipRef 속성은 각 관계 템플릿에 대한 참조를 명시한다. 서비스 템플릿이 노출하는 동작에 매핑하는 특정한 인터페이스와 동작을 각각 interfaceName 속성과 operationName 속성을 통하여 명시한다.

  비고: 참조한 관계 템플릿의 type 속성에서 정의한 관계 유형은 인터페이스와 특정한 이름의 동작을 정의하여야 한다.

- Plan: 이 요소는 서비스 템플릿이 노출하는 동작의 구현을 규정하는 계획에 대한 참조를 planRef 속성을 통하여 명시한다. Operation 요소에서 NodeOperation, RelationshipOperation 또는 Plan 중 하나를 명시하여야 한다.

- TopologyTemplate: 이 요소는 서비스 템플릿이 정의하는 클라우드 애플리케이션의 전체적인 구조(즉 이를 구성하는 컴포넌트 및 이런 컴포넌트 사이의 관계)를 명시한다. 서비스의 컴포넌트는 노드 템플릿이라고 불리며 컴포넌트 사이의 관계는 관계 템플릿이라고 불린다. TopologyTemplate 요소는 다음 특성을 가진다.

○ NodeTemplate: 이 요소는 클라우드 애플리케이션을 구성하는 컴포넌트의 종류를 명시한다. NodeTemplate 요소는 다음 특성을 가진다.

- id: 이 속성은 노드 템플릿의 식별자를 명시한다. 노드 템플릿의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- name: 이 선택적인(OPTIONAL) 요소는 노드 템플릿의 이름을 명시한다.

- type: 이 속성의 QName 값은 노드 템플릿의 유형을 규정하는 노드 유형을 참조한다.

  비고: 노드 템플릿의 type 속성이 참조하는 노드 유형을 추상적이라고 선언하는 경우, 특정한 노드 템플릿의 인스턴스를 만들 수 없다. 대신에 늦어도 노드 템플릿의 인스턴스를 만드는 동안 전문적이고 파생된 노드 유형이 있는 노드 템플릿으로 노드 템플릿을 대체하여야 한다.

- minInstances: 정수 속성은 노드 템플릿의 인스턴스를 만들 때 만들어야 하는 최소 인스턴스 수를 명시한다. 이 속성의 기본 값은 1이다. minInstances의 값은 1보다 작으면 안 된다.(MUST NOT).

- maxInstances: 이 속성은 노드 템플릿의 인스턴스를 만들 때 만들어야 하는 최대 인스턴스 수를 명시한다. 이 속성의 기본 값은 1이다(MUST). 문자열을 "unbounded"로 정한 경우, 인스턴스를 무한대로 만들 수 있다. maxInstances의 값은 1 이상이어야 하며 어떠한 경우에도 minInstances에서 명시한 값보다 더 작을 수는 없다(MUST NOT).

- Properties: 노드 템플릿의 구체적인 맥락에서 특성을 정의하는 노드 유형의 하나 이상의 특성에 대한 초기값을 명시한다. 상응하는 노드 유형 특성에 대한 XML 스키마의 인스턴스 문서를 규정함으로써 초기값을 명시한다. 이 인스턴스 문서는 노드 템플릿의 type 속성이 참조하는 노드 유형의 DerivedFrom 특성에서 비롯하는 상속 구조를 고려한다. 해당하는 XML 스키마에 제약 사항이 존재하는 경우, XML 스키마의 인스턴스 문서를 인증할 수 없다. 모든 노드 유형 특성에 초기값을 할당할 수 없을 수 있다. 즉 Properties 요소가 규정하는 인스턴스에서 의무적인 요소나 속성을 누락할 수 있다. 정의한 노드 템플릿에서 인스턴스를 만드는 경우, 관련 XML 스키마 정의에 따라 노드 유형 속성의 XML 표현을 인증한다(MUST).

- PropertyConstraints: 노드 템플릿의 특성 정의를 규정하는 노드 유형에 대하여 하나 이상의 노드 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 PropertyConstraint 요소를 통하여 각 제약 사항을 명시한다. PropertyConstraint 요소는 다음 특성을 가진다.

    - property: 이 특성의 문자열 값은 노드 유형 특성 문서(노드 템플릿의 맥락에서 제한된)에서 특성을 가리키는 XPath 표현식이다. 하나 이상의 제약조건은 각 특성에 대하여 정의되어서는 안 된다(MUST NOT).

    - constraintType: URI를(제약 사항의 의미와 콘텐츠의 포맷을 정의하는) 통하여 제약 사항 유형을 명시한다. 예를 들어 http://www.example.com/PropertyConstraints/unique의 제약 사항 유형은 정의에 따른 노드 템플릿의 참조 특성이 특정 범위 내에서 고유하여야 한다는 것을 의미할 수 있다. 각 PropertyConstraint 요소의 제약 사항 유형은 고유함을 보장하여야 하는 실제 범위를 더 상세하게 정의할 수 있다.

- Requirements: 이 요소는 노드 템플릿의 type 속성에서 명시하는 노드 유형의 요구사항 정의 목록에 따라 노드 템플릿에 대한 요구사항의 목록을 포함한다. 별도의 내포된 Requirement 요소에서 각 요구사항을 명시한다. Requirement 요소는 다음 특성을 가진다.

    - id: 이 속성은 요구사항의 식별자를 명시한다. 요구사항의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

    - name: 이 속성은 요구사항의 이름을 명시한다. 요구사항의 name 및 type은 노드 템플릿의 유형 속성에서 명시하는 노드 유형에 있는 요구사항 정의의 name 및 type과 일치하여야 한다(MUST).

    - type: 이 속성의 QName 값은 요구사항의 요구사항 유형 정의를 참조한다. 이 요구사항 유형은 요구사항의 의미론과 잠재적인 특성을 의미한다.

    - Properties: 이 요소는 특성 정의를 규정하는 요구사항 유형에 따라 하나 이상의 요구사항 특성에 대하여 초기 값을 명시한다. XML 프래그먼트의 형식으로 특성을 규정한다. 노드 템플릿의 Properties 요소에 대하여 규정한 규칙과 같은 규칙을 적용한다.

    - PropertyConstraints: 이 요소는 요구사항에 대한 특성 정의를 규정하는 하나 이상의 요구사항 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 PropertyConstraint 요소를 통하여 각 제약 사항을 명시한다. 노드 템플릿의 PropertyConstraints 요소에 대하여 규정한 규칙과 같은 규칙을 적용한다.

- Capabilities: 이 요소는 노드 템플릿의 type 속성에서 명시하는 노드 유형의 기능 정의 목록에 따라 노드 템플릿에 대한 기능의 목록을 포함한다. 별도의 내포된 Requirement 요소에서 각 요구사항을 명시한다. 별도의 내포된 Capability 요소에서 각 기능을 명시한다. Capability 요소는 다음 특성을 가진다.

    - id: 이 속성은 기능의 식별자를 명시한다. 기능의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

    - name: 이 속성은 기능의 이름을 명시한다. 기능의 name 및 type은 노드 템플릿의 유형 속성에서 명시하는 노드 유형에 있는 기능 정의의 name 및 type과 일치하여야 한다(MUST).

    - type: 이 속성의 QName 값은 기능의 기능 유형 정의를 참조한다. 이 기능 유형은 기능의 의미론과 잠재적인 특성을 의미한다.

    - Properties: 이 요소는 특성 정의를 규정하는 기능 유형에 따라 하나 이상의 기능 특성에 대하여 초기 값을 명시한다. XML 프래그먼트의 형식으로 특성을 규정한다. 노드 템플릿의 Properties 요소에 대하여 규정한 규칙과 같은 규칙을 적용한다.

    - PropertyConstraints: 이 요소는 기능에 대한 특성 정의를 규정하는 하나 이상의 기능 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 PropertyConstraint 요소를 통하여 각 제약 사항을 명시한다. 노드 템플릿의 PropertyConstraints 요소에 대하여 규정한 규칙과 같은 규칙을 적용한다.

- Policies: 이 선택적인(OPTIONAL) 요소는 노드 템플릿에 관한 정책을 명시한다. Policies 요소에서 정의하는 모든 정책은 TOSCA 구현을

통하여 실시하여야 한다(즉 정책을 AND로 결합한다). 별도의 내포된 Policy 요소를 통하여 각 정책을 정의한다. Policy 요소는 다음 특성을 가진다.

- name: 이 선택적(OPTIONAL) 속성을 통하여 정책에 대한 이름을 정의할 수 있다. 이 이름은 Policies 요소에서 고유하여야 한다.

- policyType: 이 속성은 이 정책의 유형을 명시한다. 이 속성의 QName 값은 같은 정의 문서나 불러온 문서에서 정의한 PolicyType의 QName과 일치하여야 한다(SHOULD). policyType 속성은 Policy 요소에 고유한 아티팩트 유형을 명시하고 정책이 policyRef 속성을 통하여 참조하는 정책 템플릿의 유형을 보여준다.

- policyRef: 이 선택적인(OPTIONAL) 속성의 QName 값은 노드 템플릿과 관련이 있는 정책 템플릿을 참조한다. 같은 TOSCA 정의 문서나 별도의 문서에서(현재 정의 문서로 불러온) 이 정책 템플릿을 정의할 수 있다. policyRef 속성이 참조하는 정책 템플릿의 유형은 policyType 속성에서 명시한 유형과 같은 유형이거나 그 서브 유형이어야 한다.

  비고: 정책 템플릿을 참조하지 아니하는 경우, Policy 요소의 특정 콘텐츠가 노드 템플릿의 맥락에서 충분한 정책 정보를 표현한다고 추정한다.

  비고: 정책 템플릿이 비기능적인 행동에 대한 불변 정보(즉 사용 가능성 정책의 사용 가능성 클래스와 같이 맥락과 상관이 없는 정보)를 제공하는 반면에 노드 템플릿에서 정의하는 Policy 요소는 Policy 요소의 특정 정책 본문에서 가변 정보(즉 서비스의 사용 가능성을 점검하기 위한 특정한 프레임워크 주파수와 같이 맥락에 고유한 정보)를 제공할 수 있다.

▪ DeploymentArtifacts: 이 요소는 노드 템플릿과 관련이 있는 배포 아티팩트를 명시한다. 내포된 DeploymentArtifact 요소는 개별 배포 아티팩트에 대한 세부 사항을 명시한다. DeploymentArtifact 요소는 다음 특성을 포함한다.

- name: 이 속성은 아티팩트의 이름을 명시한다. 정의를 통하여 노드 템플릿의 범위 내에서 이름이 고유할 것을 보장해야

한다(SHOULD).

- artifactType: 이 속성은 아티팩트의 유형을 명시한다. 이 속성의 QName 값은 같은 정의 문서나 불러온 문서에서 정의한 ArtifactType의 QName과 일치하여야 한다(SHOULD). ArtifactType 속성은 DeploymentArtifact 요소 본문의 콘텐츠에 특정한 아티팩트 유형을 명시하고 배포 아티팩트가 artifactRef 속성을 통하여 참조하는 아티팩트 템플릿의 유형을 보여준다.

- artifactRef: 이 선택적인(OPTIONAL) 속성은 배포 아티팩트로써 사용할 아티팩트 템플릿을 확인하는 QName을 포함한다. 이 아티팩트 템플릿을 같은 정의 문서나 별도의 내포된 문서에서 정의할 수 있다. artifactRef 속성이 참조하는 아티팩트 템플릿의 유형은 artifactType 속성에서 명시한 유형이나 유형의 서브 유형과 같아야 한다. 노드 템플릿의 type 속성 값으로써 노드 유형을 구현하는 노드 유형 구현을 사용하여 명시하는 name 및 artifactType(또는 artifactType이 파생하는 아티팩트 유형)가 같은 배포 아티팩트에 노드 템플릿에서 명시한 배포 아티팩트가 우선한다는 것에 주의한다. 그렇지 아니하는 경우, 노드 유형 배포의 배포 아티팩트와 노드 템플릿을 사용해서 정의하는 배포 아티팩트를 결합한다.

  비고: 아티팩트 템플릿을 참조하지 아니하는 경우, DeploymentArtifact 요소의 콘텐츠에 특정한 아티팩트 유형이 실제 아티팩트를 표현한다고 가정한다. 예를 들어, DeploymentArtifact 요소에서 간단한 구성 파일의 내용을 정의할 수 있다.

  노드 템플릿의 type 속성 값으로써 노드 유형을 구현하는 노드 유형 구현을 사용하여 명시하는 name 및 artifactType(또는 artifactType이 파생하는 아티팩트 유형)가 같은 배치 아티팩트에 노드 템플릿에서 명시한 배치 아티팩트가 우선한다는 것에 주의한다. 그렇지 아니하는 경우, 노드 유형 배치의 배치 아티팩트와 노드 템플릿을 사용해서 정의하는 배치 아티팩트를 결합한다.

○ RelationshipTemplate: 이 요소는 클라우드 애플리케이션 컴포넌트 사이의 관계의 종류를 명시한다. 명시한 각 관계 템플릿에 대하여 출처 요소와 목표 요소를 토폴로지 템플릿에서 명시하여야 한다(MUST). RelationshipTemplate

요소는 다음 특성을 가진다.

- **id:** 이 속성은 관계 템플릿의 식별자를 명시한다. 관계 템플릿의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- **name:** 이 선택적(OPTIONAL) 속성을 통하여 관계 템플릿의 이름을 명시한다.

- **type:** 이 속성의 QName 값은 관계 템플릿의 유형을 규정하는 관계 유형을 참조한다.

  비고: 관계 템플릿의 type 속성이 참조하는 관계 유형을 추상적이라고 선언하는 경우, 특정한 관계 템플릿의 인스턴스를 만들 수 없다. 대신에 늦어도 관계 템플릿의 인스턴스를 만드는 동안 전문적이고 파생된 관계 유형이 있는 관계 템플릿으로 관계 템플릿을 대체하여야 한다.

- **Properties:** 관계 템플릿의 구체적인 맥락에서 특성을 정의하는 관계 유형의 하나 이상의 관계 유형 특성에 대한 초기 값을 명시한다. 상응하는 관계 유형 특성에 대한 XML 스키마의 인스턴스 문서를 규정함으로써 초기 값을 명시한다. 이 인스턴스 문서는 관계 템플릿의 type 속성이 참조하는 관계 유형의 **DerivedFrom** 특성에서 비롯하는 상속 구조를 고려한다. 해당하는 XML 스키마에 제약 사항이 존재하는 경우, XML 스키마의 인스턴스 문서를 인증할 수 없다. 모든 관계 유형 특성에 초기 값을 할당할 수 없을 수 있다. 즉 **Properties** 요소가 규정하는 인스턴스에서 의무적인 요소나 속성을 누락할 수 있다. 정의한 노드 템플릿에서 인스턴스를 만드는 경우, 관련 XML 스키마 정의에 따라 관계 유형 속성의 XML 표현을 인증한다(MUST).

- **PropertyConstraints:** 관계 템플릿의 특성 정의를 규정하는 관계 유형에 대하여 하나 이상의 관계 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 **PropertyConstraint** 요소를 통하여 각 제약 사항을 명시한다. **PropertyConstraint** 요소는 다음 특성을 가진다.

  - **property:** 이 특성의 문자열 값은 관계 유형 특성 문서(관계 템플릿의 맥락에서 제한된)에서 특성을 가리키는 XPath 표현식이다. 하나 이상의 제약 사항은 각 특성에 대하여 정의되어서는 안 된다(MUST NOT).

  - **constraintType:** URI를(제약 사항의 의미와 콘텐츠의 포맷을 정의하는) 통하여 제약 사항 유형을 명시한다. 예를 들어 http://www.example.com/PropertyConstraints/unique의 제약 사항 유형은 정의에 따른 노드 템플릿의 참조 특성이 특정 범위 내에서 고유하여야 한다는 것을 의미할 수 있다. 각 **PropertyConstraint** 요소의 제약 사항 유형은 고유함을 보장하여야 하는 실제 범위를 더 상세하게 정의할 수 있다.

- **SourceElement:** 이 요소는 현재의 관계 템플릿이 의미하는 관계의 기점을 명시한다. **SourceElement** 요소는 다음 특성을 가진다.

  - **ref:** 이 속성은 관계 템플릿의 출처인 같은 서비스 템플릿 문서에서 ID를 통하여 노드 템플릿이나 노드 템플릿의 요구사항을 참조한다. type 속성이 참조하는 관계 유형이 **ValidSource** 요소를 통하여 관계의 유효한 출처에 대한 제약 사항을 정의하는 경우, **SourceElement**의 ref 요소는 객체를(유형이 각 관계 유형의 유효한 출처 제약 사항을 준수하는) 참조하여야 한다(MUST). 노드 유형을 관계 유형 정의에서 유효한 출처로써 정의하는 경우, **ref** 속성은 해당하는 노드 유형(또는 그 서브 유형)의 노드 템플릿을 참조하여야 한다. 요구사항 유형을 관계 유형에 있는 유효한 출처로써 정의하는 경우, **ref** 속성은 노드 템플릿에 있는 해당 요구사항 유형의 요구사항을 참조하여야 한다.

- **TargetElement:** 이 요소는 현재의 관계 템플릿이 의미하는 관계의 목표를 명시한다. **TargetElement** 요소는 다음 특성을 가진다.

  - **ref:** 이 속성은 관계 템플릿의 목표인 같은 서비스 템플릿 문서에서 ID를 통하여 노드 템플릿이나 노드 템플릿의 기능을 참조한다.

    type 속성이 참조하는 관계 유형이 **ValidTarget** 요소를 통하여 관계의 유효한 출처에 대한 제약 사항을 정의하는 경우, **TargetElement**의 **ref** 요소는 객체를(유형이 각 관계 유형의 유효한 출처 제약 사항을 준수하는) 참조하여야 한다.

    노드 유형을 관계 유형 정의에서 유효한 목표로써 정의하는 경우, **ref** 속성은 해당하는 노드 유형(또는 그 서브 유형)의 노드 템플릿을 참조하여야 한다(MUST).

기능 유형을 관계 유형에 있는 유효한 목표로써 정의하는 경우, ref 속성은 노드 템플릿에 있는 해당 기능 유형의 기능을 참조하여야 한다(MUST).

- RelationshipConstraints: 이 요소는 관계의 사용에 대한 제약 사항의 목록을 별도의 내포된 RelationshipConstraint 요소에서 명시한다. RelationshipConstraint 요소는 다음 특성을 가진다.

  - constraintType: 이 속성은 URI를 통하여 관계 제약 사항의 유형을 명시한다. 유형에 따라 RelationshipConstraint 요소는 유형에 고유한 콘텐츠(실제 제약 사항을 더욱 상세하게 설명하는)를 포함할 수 있다.

  - Plans: 이 요소는 서비스의 동작 활동을 명시한다. 계획 요소에 포함된 Plan은 서비스를 만들거나 종료하거나 관리하는 방법을 명시할 수 있다. Plan 요소는 다음 특성을 가진다.

○ id: 이 속성은 계획의 식별자를 명시한다. 계획의 식별자는 목표 네임스페이스에서 고유하여야 한다.

○ name: 이 선택적인(OPTIONAL) 속성은 계획의 이름을 명시한다.

○ planType: 속성의 값은 계획의 실행이 서비스에 미치는 영향에 대한 징후로써 계획의 유형을 명시한다. URI를 통하여 계획 유형을 명시하여 서비스 템플릿을 작성하는 자가 시간의 흐름에 따라 새로운 계획 유형을 정의할 수 있도록 한다. TOSCA 설명서의 일부로써 다음 계획 유형을 정의한다.

- http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan – 이 URI는 서비스 템플릿으로부터 서비스의 새로운 인스턴스를 처음으로 만드는데 사용하는 계획에 대한 계획 유형(빌드 계획)을 정의한다.

- http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan – 이 URI는 서비스 인스턴스를 종료하는 데 사용하는 계획에 대한 계획 유형(종료 계획)을 정의한다.

서비스 인스턴스의 수명 주기 동안 서비스 인스턴스를 관리하기 위한 다른 모든 계획 유형을 고려하고 이를 일반적으로 변경 계획이라고 부르는

것에 주의한다.

○ planLanguage: 이 속성은 계획을 명시하는 프로세스 모델링 언어(또는 메타 모델)를 의미한다. 예를 들어 http://www.omg.org/spec/BPMN/20100524/MODEL은 계획을 모델화하는 데 BPMN 2.0을 사용하였다는 것을 명시할 것이다.

TOSCA는 계획을 정의하기 위하여 별도의 메타 모델을 명시하지 아니한다. 대신에 BPEL [BPEL 2.0]이나 BPMN [BPMN 2.0]과 같은 프로세스 모델링 언어(메타 모델로도 알려진)을 사용하여 계획을 정의할 것으로 가정한다. The 설명서는 모델링 계획으로써 BPMN을 사용하는 것을 선호한다.

○ Precondition: 이 선택적인(OPTIONAL) 요소는 계획을 실시하기 위하여 충족하여야 하는 조건을 명시한다. 이 요소의 expressionLanguage 속성은 내포된 조건을 규정하는 표현 언어를 명시한다.

일반적으로 전제 조건은 토폴로지 템플릿의 노드 템플릿이나 관계 템플릿 중 일부의 인스턴스 상태 속성으로 표현한다. 계획을 실시할 것을 요청하는 시점에 해당 속성의 실제 값에 기초하여 평가한다. 모든 종류의 전제 조건이 허용된다는 것에 주의한다.

○ InputParameters: 이 선택적인(OPTIONAL) 특성은 계획에 대한 하나 이상의 입력 매개 변수 정의의 목록을 포함한다. 각 입력 매개 변수의 정의를 내포된 별도의 InputParameter 요소로 정의한다. InputParameter 요소는 다음 특성을 가진다.

- name: 이 속성은 입력 매개 변수의 이름을 명시한다. 이는 동작을 위하여 정의한 일련의 입력 매개 변수에서 고유하여야 한다.

- type: 이 속성은 입력 매개 변수의 유형을 명시한다.

- required: 이 선택적인(OPTIONAL) 특성은 입력 매개 변수가 필수 사항(required 속성의 값이 “예”인 경우. 기본 값)인지 아니면 선택 사항(required 속성의 값이 “아니오”인 경우)인지를 명시한다.

○ OutputParameters: 이 선택적인(OPTIONAL) 속성은 계획에 대한 하나 이상의 출력 매개 변수 정의의 목록을 포함한다. 각 출력 매개 변수의 정의를 내포된 별도의 OutputParameter 요소로 정의한다. OutputParameter 요소는 다음 특성을 가진다.

- **name:** 이 속성은 출력 매개 변수의 이름을 명시한다. 이는 동작을 위하여 정의한 일련의 출력 매개 변수에서 고유하여야 한다.

- **type:** 이 속성은 출력 매개 변수의 유형을 명시한다.

- **required:** 이 선택적인(OPTIONAL) 속성은 출력 매개 변수가 필수 사항(**required** 속성의 값이 "예"인 경우. 기본 값)인지 아니면 선택 사항(**required** 속성의 값이 "아니오"인 경우)인지를 명시한다.

○ **PlanModel:** 이 특성은 실제 모델 콘텐츠를 포함한다.

○ **PlanModelReference:** 이 특성은 모델 콘텐츠를 가리킨다. 속성에 대한 참조는 계획의 모델 URI를 포함한다.

Plan 요소의 인스턴스는 **PlanModel** 요소의 인스턴스로써 실제 계획을 포함하거나 **PlanModelReference** 요소를 통하여 모델을 가리켜야 한다(MUST).

9.3 예시

다음 서비스 템플릿은 "MyApplication" 및 "MyAppServer"라고 불리는 2개의 노드 템플릿을 포함하는 토폴로지 템플릿을 정의한다. 이런 노드 템플릿은 "Application" 및 "ApplicationServer"라고 불리는 노드 유형을 가진다. "MyApplication" 노드 템플릿을 정확히 한 번 인스턴스로 만든다. 해당 **Properties** 요소가 2개의 노드 유형 특성을 초기화한다. "MyAppServer" 노드 템플릿은 필요한 경우 여러 번 인스턴스로 만들 수 있다. "MyHostedRelationship"이라는 이름의 관계 템플릿을 통하여 "MyApplication" 노드 템플릿을 "MyAppServer" 노드 템플릿과 연결한다. 관계 템플릿의 행동과 의미론을 관계 유형 "HostedOn"에서 정의한다. 즉 "MyApplication"을 "MyAppServer"에 호스트 한다. 또한 서비스 템플릿은 애플리케이션 서버에 호스트 한 "MyApplication" 애플리케이션을 업데이트하기 위하여 "UpdateApplication" 계획을 정의한다. 이 계획은 별도의 파일에 있는 BPMN 2.0 프로세스 정의를 참조한다.

```
1  <ServiceTemplate id="MyService"
2                   name="My Service">
3
4    <TopologyTemplate>
5
6      <NodeTemplate id="MyApplication"
7                    name="My Application"
8                    type="my:Application">
9        <Properties>
10         <ApplicationProperties>
11           <Owner>Frank</Owner>
```



```
12           <InstanceName>Thomas' favorite application</InstanceName>
13         </ApplicationProperties>
14       </Properties>
15     </NodeTemplate>
16
17     <NodeTemplate id="MyAppServer"
18                   name="My Application Server"
19                   type="my:ApplicationServer"
20                   minInstances="0"
21                   maxInstances="unbounded"/>
22
23     <RelationshipTemplate id="MyDeploymentRelationship"
24                           type="my:deployedOn">
25       <SourceElement ref="MyApplication"/>
26       <TargetElement ref="MyAppServer"/>
27     </RelationshipTemplate>
28
29   </TopologyTemplate>
30
31   <Plans>
32     <Plan id="UpdateApplication"
33        planType="http://www.example.com/UpdatePlan"
34        planLanguage="http://www.omg.org/spec/BPMN/20100524/MODEL">
35       <PlanModelReference reference="plans:UpdateApp"/>
36     </Plan>
37   </Plans>
38
39 </ServiceTemplate>
```

10 노드 유형

이 절은 *노드 유형*을 정의하는 방법을 명시한다. 노드 유형은 다시 사용할 수 있는 엔티티(하나 이상의 노드 템플릿 유형을 정의하는)이다. 노드 유형은 *특성 정의*(즉 노드 유형이나 이런 노드 템플릿의 인스턴스를 사용하여 노드 템플릿에서 정의한 특성의 이름, 데이터 유형 및 허용 값)를 통하여 가시적인 특성의 구조를 정의한다.

노드 유형은 **DerivedFrom** 요소를 통하여 다른 노드 유형으로부터 특성을 상속할 수 있다. 노드 유형을 추상적이라고 선언할 수 있다. 이는 노드 유형을 인스턴스로 만들 수 없다는 것을 의미한다. 추상적인 노드 유형의 목적은 전문적이고 파생된 노드 유형에서 다시 사용하기 위하여 공통의 특성과 행동을 규정하는 것이다. 또한 노드 유형을 최종적인 것으로 선언할 수도 있다. 이는 다른 노드 유형으로부터 파생될 수 없다는 것을 의미한다.

노드 유형은 각각 **RequirementDefinition** 요소나 **CapabilityDefinition** 요소를 통하여 특정 요구사항과 기능(제7.4절 참조)을 노출할 것을 선언할 수 있다.

해당 노드 템플릿(인스턴스)에서 실시할 수 있는 함수를 노드 유형의 인터페이스가 정의한다. 마지막으로 노드 유형에 대하여 관리 정책을 정의한다.

10.1 XML 구문

다음 의사 스키마는 노드 유형의 XML 문법을 정의한다.

```
<NodeType name="xs:NCName" targetNamespace="xs:anyURI"?
        abstract="yes|no"?final="yes|no"?>

  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

  <DerivedFrom typeRef="xs:QName"/> ?

  <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

  <RequirementDefinitions>
    <RequirementDefinition name="xs:string"
                          requirementType="xs:QName"
                          lowerBound="xs:integer"?
                          upperBound="xs:integer | xs:string"?>
      <Constraints>
        <Constraint constraintType="xs:anyURI">
          constraint type specific content
        </Constraint> +
      </Constraints> ?
    </RequirementDefinition> +
  </RequirementDefinitions> ?

  <CapabilityDefinitions>
    <CapabilityDefinition name="xs:string"
                         capabilityType="xs:QName"
                         lowerBound="xs:integer"?
                         upperBound="xs:integer | xs:string"?>
      <Constraints>
        <Constraint constraintType="xs:anyURI">
          constraint type specific content
        </Constraint> +
      </Constraints> ?
    </CapabilityDefinition> +
  </CapabilityDefinitions>

  <InstanceStates>
    <InstanceState state="xs:anyURI"> +
  </InstanceStates> ?

  <Interfaces>
    <Interface name="xs:NCName| xs:anyURI">
      <Operation name="xs:NCName">
        <InputParameters>
          <InputParameter name="xs:string" type="xs:string"
                         required="yes|no"?/>+
        </InputParameters> ?
        <OutputParameters>
          <OutputParameter name="xs:string" type="xs:string"
                          required="yes|no"?/>+
        </OutputParameters> ?
      </Operation> +
    </Interface> +
  </Interfaces> ?

</NodeType>
```


10.2 속성

NodeType 요소는 다음 특성을 가진다.

- name: 이 속성은 노드 유형의 이름이나 식별자를 명시한다. 노드 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- targetNamespace: 이 선택적인(OPTIONAL) 속성은 노드 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 노드 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다.

- abstract: 이 선택적인(OPTIONAL) 속성은 노드 유형을 유형으로써 사용하는 노드 템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다. 노드 유형이 추상적인 요구사항 유형이나 기능 유형의 요구사항 정의나 기능 정의를 포함하는 경우, 노드 유형을 추상적인 것으로 선언하여야 한다(MUST).

  결과적으로 늦어도 노드 템플릿을 인스턴스로 만드는 동안 노드 템플릿이 참조하는 추상적인 노드 유형을 추상적인 노드 유형에서 파생된 노드 유형으로 대체하여야 한다.

  비고: 추상적인 노드 유형을 최종적인 것으로 선언해서는 안 된다.

- final: 이 선택적인(OPTIONAL) 속성은 이 노드 유형에서 다른 노드 유형을 파생하여서는 아니 된다는 것을 명시한다.

  비고: 최종 노드 유형을 추상적인 것으로 선언해서는 안 된다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(노드 유형을 작성하는 자가 노드 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  - name: 이 속성은 태그의 이름을 명시한다.

  - value: 이 속성은 태그의 값을 명시한다.

비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- DerivedFrom: 이는 이 노드 유형이 파생하는 다른 노드 유형을 참조하는 선택적인 요소이다. 로컬 정의가 파생된 정의에 항상 우선한다는 규칙에 따라 서로 모순되는 정의를 해결한다. 자세한 사항에 대해서는 제10.3절 파생 규칙을 참조한다. DerivedFrom 요소는 다음 특성을 가진다.
    - typeRef: QName은 이 노드 유형의 정의가 파생된 출처 노드 유형을 명시한다.
- PropertiesDefinition: 이 요소는 XML 스키마를 통하여 노드 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. PropertiesDefinition 요소는 다음 특성 중 하나의 특성만을 가진다.
    - element: 이 속성은 노드 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.
    - type: 이 속성은 노드 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.
- RequirementDefinitions: 이 선택적인(OPTIONAL) 요소는 노드 유형이 노출하는 요구사항을 명시한다(자세한 사항에 대해서는 제7.4절을 참조한다). 내포된 RequirementDefinition 요소에서 각 요구사항을 정의한다. RequirementDefinition 요소는 다음 특성을 가진다.
    - name: 이 속성은 정의한 요구사항의 이름을 명시하며 현재 노드 유형의 RequirementsDefinitions에서 고유하여야 한다(MUST).

      하나의 노드 유형이 같은 요구사항 유형에 대한 복수의 요구사항을 정의할 수 있다는 것에(이 경우 이름을 통하여 각 요구사항 정의를 고유하게 식별한다) 주의한다. 예를 들어 애플리케이션에 대한 노드 유형은 데이터베이스에 대하여 두 가지의 요구사항(즉 간은 요구사항 유형의)을 정의할 수 있다. 하나의 요구사항을 “customerDatabase”로 명명할 수 있고 다른 요구사항을 “productsDatabase”로 명명할 수 있다.
    - requirementType: 이 속성은 QName을 통하여 요구사항 유형(현재의 RequirementDefinition이 정의하는)을 확인하다.

    - lowerBound: 이 선택적인(OPTIONAL) 속성은 하위 경계를 명시한다. 이를 통하여 요구사항은 현재 노드 유형(예를 들어 노드 템플릿에 대하여 만든 노드 유형)에 따라 노드 템플릿과 매칭 하여야 한다. 이 속성에 대한 기본 값은 1이다. 이 속성의 값이 0인 경우, 이는 요구사항의 매칭이 선택 사항임을 의미한다.
    - upperBound: 이 선택적인(OPTIONAL) 속성은 상위 경계를 명시한다. 이를 통하여 요구사항은 현재 노드 유형(예를 들어 노드 템플릿에 대하여 만든 노드 유형)에 따라 노드 템플릿과 매칭 하여야 한다. 이 속성에 대한 기본 값은 1이다. 이 속성의 값이 “unbounded”인 경우, 이는 상위 경계가 없음을 의미한다.
    - Constraints: 이 선택적인(OPTIONAL) 요소는 요구사항 정의에 대한 추가적인 제약 사항을 명시하는 Constraint 요소의 목록을 포함한다. 예를 들어 데이터베이스가 필요로 하는 경우, 지원하는 SQL 특징에 대한 제약 사항을 표현할 수 있다. 내포된 Constraint 요소는 다음 특성을 가진다.
        - constraintType: 이 속성은 제약 사항의 유형을 명시한다. 이 유형에 따르면 Constraint 요소는 유형에 고유한 콘텐츠를 포함한다.
- CapabilityDefinitions: 이 선택적인(OPTIONAL) 요소는 노드 유형이 노출하는 기능을 명시한다. 내포된 CapabilityDefinition 요소에서 각 기능을 정의한다. CapabilityDefinition 요소는 다음 특성을 가진다.
    - name: 이 속성은 정의한 기능의 이름을 명시하며 현재 노드 유형의 CapabilityDefinitions에서 고유하여야 한다(MUST).

      하나의 노드 유형이 같은 요구사항 유형에 대한 복수의 요구사항을 정의할 수 있다는 것에(이 경우 이름을 통하여 각 요구사항 정의를 고유하게 식별한다) 주의한다.
    - capabilityType: 이 속성은 QName을 통하여 기능 유형(현재의 CapabilityDefinition이 정의하는)을 확인하다.
    - lowerBound: 이 선택적인(OPTIONAL) 속성은 정의할 기능의 대상인 필수 노드의 하위 경계를 명시한다. 이 속성에 대한 기본 값은 1이다. 이 속성의 값이 0인 경우, 이는 기능이 필수 노드를 실제로 충족할 수 없다는 것을 의미하기 때문에 유효하지 아니하다.
    - upperBound: 이 선택적인(OPTIONAL) 속성은 정의한 기능의 대상인

클라이언트 요구사항의 상위 경계를 명시한다. 이 속성에 대한 기본 값은 1이다. 이 속성의 값이 “unbounded”인 경우, 이는 상위 경계가 없음을 의미한다.

- Constraints: 이 선택적인(OPTIONAL) 요소는 기능 정의에 대한 추가적인 제약 사항을 명시하는 Constraint 요소의 목록을 포함한다. 내포된 Constraint 요소는 다음 특성을 가진다.
  - constraintType: 이 속성은 제약 사항의 유형을 명시한다. 이 유형에 따르면 Constraint 요소는 유형에 고유한 콘텐츠를 포함한다.

- InstanceStates: 이 선택적인(OPTIONAL) 요소는 이 노드 유형에 대한 일련의 상태를 나열한다. 내포된 InstanceState 요소에서 이런 상태를 정의한다. InstanceState 요소는 다음과 같은 내포된 특성을 가진다.
  - state: 이 속성은 잠재적인 상태를 확인하는 URI를 명시한다.

- Interfaces: 이 요소는 이 노드 유형(인스턴스)에서 실시할 수 있는 동작의 정의를 포함한다. 이런 동작을 내포된 Interface 요소의 형태로 정의한다. Interface 요소는 다음 특성을 가진다.
  - name: 인터페이스의 이름. 이 이름은 URI이거나 NCName이며 정의 대상인 노드 유형의 범위에서 고유하여야 한다(MUST).
  - Operation: 이 요소는 노드 유형의 특정 측면을 관리하는 데 사용할 수 있는 동작을 정의한다. Operation 요소는 다음 특성을 가진다.
    - name: 이 속성은 동작의 이름을 명시하며 노드 유형의 인터페이스에서 고유하여야 한다(MUST).
    - InputParameters: 이 선택적인(OPTIONAL) 특성은 하나 이상의 입력 매개 변수 정의의 목록을 포함하며 각 입력 매개 변수를 내포된 별도의 InputParameter 요소에서 정의한다. InputParameter 요소는 다음 특성을 가진다.
      - name: 이 속성은 입력 매개 변수의 이름을 명시하며 동작에 대하여 정의한 일련의 입력 매개 변수에서 고유하여야 한다(MUST).
      - type: 이 속성은 입력 매개 변수의 유형을 명시한다.

      - required: 이 선택적인(OPTIONAL) 속성은 입력 매개 변수가 필수 사항(required 속성의 값이 “예”인 경우. 기본 값)인지 아니면 선택 사항(required 속성의 값이 “아니오”인 경우)인지를 명시한다.
    - OutputParameters: 이 선택적인(OPTIONAL) 특성은 하나 이상의 출력 매개 변수 정의의 목록을 포함하며 각 출력 매개 변수를 내포된 별도의 OutputParameter 요소에서 정의한다. OutputParameter 요소는 다음 특성을 가진다.
      - name: 이 속성은 출력 매개 변수의 이름을 명시하며 동작에 대하여 정의한 일련의 출력 매개 변수에서 고유하여야 한다.
      - type: 이 속성은 출력 매개 변수의 유형을 명시한다.
      - required: 이 선택적인(OPTIONAL) 속성은 출력 매개 변수가 필수 사항(required 속성의 값이 “예”인 경우. 기본 값)인지 아니면 선택 사항(required 속성의 값이 “아니오”인 경우)인지를 명시한다.

10.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 노드 유형 특성: 노드 유형 특성을 의미하는 XML 요소(또는 유형)가 DerivedFrom 요소에서 참조하는 노드 유형의 노드 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

- 요구사항 및 기능: 정의 중인 노드 유형의 일련의 요구사항이나 기능은 파생된 노드 유형이 정의하는 요구사항이나 기능과 정의 중인 노드 유형이 정의한 기능이나 요구사항의 결합으로 구성된다.

  정의 중인 노드 유형이 특정한 이름이 있는 요구사항이나 기능을 정의하는 경우(파생된 노드 유형이 이름이 같은 각각의 정의를 이미 포함하는 경우), 정의 중인 노드 유형에 있는 정의는 파생된 노드 유형에 있는 정의에 우선한다. 이런 경우, 요구사항 정의나 기능 정의는 각각 요구사항 유형이나 기능 유형(파생된 노드 유형의 해당 요구사항 정의나 기능 정의에서 파생한)을 참조하여야 한다.

- 인스턴스 상태: 정의 중인 노드 유형의 일련의 인스턴스 상태는 파생된 노드

유형이 정의하는 인스턴스 상태와 정의 중인 노드 유형이 정의하는 인스턴스 상태의 결합으로 구성된다. 이름이 같은 일련의 인스턴스 상태를 결합하여 같은 이름의 하나의 인스턴스 상태로 만든다.

- 인터페이스: 정의 중인 노드 유형의 일련의 인터페이스는 파생된 노드 유형이 정의하는 인터페이스와 정의 중인 노드 유형이 정의하는 인터페이스의 결합으로 구성된다.
  이름이 같은 두 개의 인터페이스를 결합하여 같은 이름의 하나의 파생된 인터페이스로 만든다. 파생된 인터페이스의 여러 동작은 두 인터페이스가 정의하는 일련의 동작의 결합으로 구성된다. 정의 중인 노드 유형이 정의하는 동작은 파생된 노드 유형에서 이름이 같은 동작을 대체한다.

10.4 예시

다음 예제는 노드 유형 “Project”를 정의한다. 목표 네임스페이스 http://www.example.com/sample에 있는 정의 문서 “MyDefinitions”에서 노드 유형 “Project”를 정의한다. 따라서 다른 정의 문서에 있는 해당 네임스페이스를 불러와서 프로젝트 노드 유형을 다른 문서에서 사용할 수 있다.

```
<Definitions id="MyDefinitions" name="My Definitions"
             targetNamespace="http://www.example.com/sample">

  <NodeType name="Project">

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

    <PropertiesDefinition element="ProjectProperties"/>

    <InstanceStates>
      <InstanceState state="www.example.com/active"/>
      <InstanceState state="www.example.com/onHold"/>
    </InstanceStates>

    <Interfaces>
      <Interface name="ProjectInterface">
        <Operation name="CreateProject">
          <InputParameters>
            <InputParamter name="ProjectName"
                           type="xs:string"/>
            <InputParamter name="Owner"
                           type="xs:string"/>
            <InputParamter name="AccountID"
                           type="xs:string"/>
          </InputParameters>
        </Operation>
      </Interface>
    </Interfaces>
  </NodeType>

</Definitions>
```

노드 유형 “Project”는 세 가지의 노드 유형 특성(모두 “xs:string” 유형인 Owner, ProjectName 및 AccountID)을 가지며 서비스 템플릿 문서의 Types 요소 정의에서 이를 XML 요소로써 정의한다. 노드 유형 “Project”의 인스턴스는 “활성화”(더 정확한 상태에 대해서는 “www.example.com/active”를 참조한다) 상태이거나 “보류”(더 정확한 상태에 대해서는 www.example.com/onHold를 참조한다) 상태일 수 있다. 이 노드 유형에 대하여 하나의 인터페이스를 정의한다. 동작이 이 인터페이스를 정의한다. 즉 동작을 정의함으로써 실제 구현을 정의한다. 동작의 이름은 `CreateProject`이며 세 가지의 매개 변수를 가진다(InputParameter 요소의 required 속성의 기본 값은 “예”이다). 이런 입력 매개변수의 이름은 ProjectName, Owner 및 AccountID이며 모두 “xs:string” 유형이다.

11 노드 유형 구현

이 절은 노드 유형 구현을 정의하는 방법을 명시한다. 노드 유형 구현은 특정한 노드 유형을 구현하는 실행 코드를 의미한다. 이는 노드 유형의 인터페이스 동작을 구현하는 실행 코드(구현 아티팩트라고 알려진)와 특정 노드 유형을 참조하는 노드 템플릿의 인스턴스를 실현하는 데 필요한 실행 코드(배포 아티팩트라고 알려진)의 컬렉션을 구현한다. 이런 각각의 실행 코드를 별도의 아티팩트 템플릿으로 정의하고 노드 유형 구현의 구현 아티팩트와 배포 아티팩트에서 이를 참조한다.

아티팩트 템플릿이 아티팩트에 대한 불변 정보(즉 아티팩트의 파일 이름과 같이 맥락으로부터 독립적인 정보)를 제공하는 반면에 구현 또는 배포 아티팩트는 특정 환경에 대한 인증 데이터나 배포 경로와 같은 가변 정보(또는 맥락에 고유한)를 제공할 수 있다.

노드 유형 구현은 TOSCA 컨테이너에 대한 힌트(필수 컨테이너 특징의 정의를 통하여 특정한 환경에 맞는 적절한 구현 방법을 선택할 수 있도록 하는)를 명시할 수 있다.

11.1 XML 구문

다음 의사 스키마는 노드 유형 구현의 XML 문법을 정의한다.

```
<NodeTypeImplementation name="xs:NCName" targetNamespace="xs:anyURI"?
                        nodeType="xs:QName"
                        abstract="yes|no"?
                        final="yes|no"?>

  <Tags>
```

```
7      <Tag name="xs:string" value="xs:string"/> +
8    </Tags> ?
9
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>
```

11.2 속성

NodeTypeImplementation 요소는 다음 특성을 가진다.

- name: 이 속성은 노드 유형 구현의 이름이나 식별자를 명시한다. 노드 유형 구현의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- targetNamespace: 이 선택적인(OPTIONAL) 속성은 노드 유형 구현의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 노드 유형 구현을 정의 문서의 목표 네임스페이스에 추가한다.

- nodeType: 이 속성의 QName 값은 이 노드 유형 구현이 구현하는 노드 유형을 명시한다.

- abstract: 이 선택적인(OPTIONAL) 속성은 이 노드 유형 구현을 직접 사용하여 nodeType 속성에서 명시한 노드 유형을 구현할 수 없다는 것을 명시한다.

  예를 들어 노드 유형을 구현하는 자는 다시 사용할 수 있도록 특정 노드 유형의 구현 중 일부만(즉 일부 동작만) 전달하고 특정한 동작을 더 구체적이고 파생된 노드 유형 구현으로 전달하도록 요구할 것을 결정할 수 있다.

  비고: 추상적인 노드 유형 구현을 최종적인 것으로 선언해서는 안 된다.

- final: 이 선택적인(OPTIONAL) 속성은 이 노드 유형 구현에서 다른 노드 유형 구현을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 노드 유형 구현을 추상적인 것으로 선언해서는 안 된다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(노드 유형 구현을 작성하는 자가 노드 유형 구현을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  ○ name: 이 속성은 태그의 이름을 명시한다.

  ○ value: 이 속성은 태그의 값을 명시한다.

  비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- DerivedFrom: 이는 이 노드 유형 구현이 파생하는 다른 노드 유형 구현을 참조하는 선택적인(OPTIONAL) 요소이다. 자세한 사항에 대해서는 제10.3절 파생 규칙을 참조한다. DerivedFrom 요소는 다음 특성을 가진다.

  ○ nodeTypeImplementationRef: QName은 이 노드 유형 구현이 파생되는 노드 유형 구현을 명시한다.

- RequiredContainerFeatures: TOSCA 컨테이너의 특정한 API(사유 API일 수 있는)와 같이 노드 유형을 실행하는 환경의 특정한 특징에 따라 노드 유형의 구현은 다를 수 있다. 예를 들어 이미지에 기초하여 가상 머신을 배포하기 위한 구현은 공용 클라우드가 제공하는 일부 API에 접근할 것을 요구할 수 있는 반면에 다른 구현은 판매회사 고유의 가상 이미지 라이브러리에 대한 API를 요구할 수 있다. 따라서 RequiredContainerFeatures 요소의 콘텐츠는 TOSCA 컨테이너에 "힌트"를 제공하여 복수의 대안이 있는 경우 적절한 노드 유형 구현을 선택할 수 있도록 한다. 별도의 RequiredContainerFeature 요소가 종속성을 정의한다. RequiredContainerFeature 요소는 다음 특성을 가진다.

  ○ feature: 이 속성의 값은 환경의 해당 특징을 의미하는 URI이다.

- ImplementationArtifacts: 이 요소는 노드 유형의 인터페이스나 동작에 대한

일련의 구현 아티팩트를 명시한다. ImplementationArtifacts 요소는 다음 특성을 가진다.

○ ImplementationArtifact: 이 요소는 인터페이스나 동작에 대하여 하나의 구현 아티팩트를 명시한다. ImplementationArtifact 요소는 다음 특성을 가진다.

비고: 다음에서 정의하는 속성에 따라 노드 유형을 구현하기 위하여 복수의 구현 아티팩트가 필요할 수도 있다. 구현 아티팩트는 노드 유형에 대하여 정의한 모든 인터페이스와 동작에 대한 구현으로써 역할을 할 수가 있다. 구현 아티팩트는 하나의 인터페이스(및 모든 동작)에 대한 구현으로써 역할을 하거나 하나의 특정한 동작에 대한 구현으로써 역할을 할 수가 있다.

▪ name: 이 속성은 아티팩트의 이름을 명시한다. 아티팩트의 이름은 노드 유형 구현의 범위 내에서 고유하여야 한다.

▪ interfaceName: 이 선택적인(OPTIONAL) 속성은 실제 구현 아티팩트가 구현하는 인터페이스의 이름을 명시한다. 명시하지 아니하는 경우, 구현 아티팩트가 NodeTypeImplementation의 nodeType 속성이 참조하는 노드 유형이 정의하는 모든 인터페이스를 구현한다고 가정한다.

▪ operationName: 이 선택적인(OPTIONAL) 속성은 실제 구현 아티팩트가 구현하는 동작의 이름을 명시한다. 명시하는 경우, interfaceName을 명시하고 명시한 operationName은 명시한 인터페이스의 동작을 참조하여야 한다(MUST). 명시하지 아니하는 경우, 구현 아티팩트가 명시한 인터페이스에서 정의한 모든 동작을 구현한다고 가정한다.

▪ artifactType: 이 속성은 이 아티팩트의 유형을 명시한다. 이 속성의 QName은 같은 정의 문서나 불러온 문서에서 정의한 ArtifactType의 QName과 일치하여야 한다. artifactType 속성은 ImplementationArtifact 요소에 고유한 아티팩트 유형을 명시하고 구현 아티팩트가 artifactRef 속성을 통하여 참조하는 Artifact Template의 유형을 의미한다.

▪ artifactRef: 이 선택적인(OPTIONAL) 속성은 QName을 포함한다. QName은 구현 아티팩트로써 사용할 아티팩트 템플릿을 확인한다. 같은 정의 문서나 별도의 불러온 문서에서 이 아티팩트 템플릿을 정의할 수 있다. artifactRef 속성이 참조하는 아티팩트 템플릿의 유형은 artifactType 속성에서 명시한 유형과 같은 유형이거나 그 서브 유형이어야 한다(MUST).

비고: 아티팩트 템플릿을 참조하지 아니하는 경우, ImplementationArtifact 요소에 고유한 아티팩트 유형이 실제 아티팩트를 나타내는 것으로 가정한다. 예를 들어 간단한 스크립트를 ImplementationArtifact 요소에서 정의할 수 있다.

● DeploymentArtifacts: 이 요소는 구현하는 노드 유형의 노드 인스턴스를 구체화하는 것과 관련한 일련의 배포 아티팩트를 명시한다. DeploymentArtifacts 요소는 다음 특성을 가진다.

○ DeploymentArtifact: 이 요소는 하나의 배포 아티팩트를 명시한다. DeploymentArtifact 요소는 다음 특성을 가진다.

비고: 하나의 노드 유형 구현에서 복수의 배치 아티팩트를 정의할 수 있다. 이는 전체적으로 하나의 노드를 구체화하는 데 복수의 아티팩트(아마도 다른 유형의)가 필요할 수도 있기 때문이다. 다른 이유는 다른 상황에서 사용하기 위하여 다른 아티팩트를 규정할 수도 있기 때문이다(가령 다른 운영 체계에서 사용하기 위한 소프트웨어의 다른 설치 코드).

▪ name: 이 속성은 아티팩트의 이름을 명시한다. 아티팩트의 이름은 노드 유형 구현의 범위 내에서 고유하여야 한다.

▪ artifactType: 이 속성은 이 아티팩트의 유형을 명시한다. 이 속성의 QName은 같은 정의 문서나 불러온 문서에서 정의한 ArtifactType의 QName과 일치하여야 한다. artifactType 속성은 DeploymentArtifact 요소에 고유한 아티팩트 유형을 명시하고 구현 아티팩트가 artifactRef 속성을 통하여 참조하는 Artifact Template의 유형을 의미한다.

▪ artifactRef: 이 선택적인(OPTIONAL) 속성은 QName을 포함한다. QName은 배포 아티팩트로써 사용할 아티팩트 템플릿을 확인한다. 같은 정의 문서나 별도의 불러온 문서에서 이 아티팩트 템플릿을 정의할 수 있다. artifactRef 속성이 참조하는 아티팩트 템플릿의 유형은 artifactType 속성에서 명시한 유형과 같은 유형이거나 그 서브 유형이어야 한다(MUST).

비고: 아티팩트 템플릿을 참조하지 아니하는 경우, DeploymentArtifact 요소에 고유한 아티팩트 유형이 실제 아티팩트를 나타내는 것으로 가정한다. 예를 들어 간단한 구성 파일을 DeploymentArtifact 요소에서 정의할 수 있다.

11.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 구현 아티팩트: 노드 유형 구현의 일련의 구현 아티팩트는 노드 유형 구현 자체가 정의하는 구현 아티팩트와 파생한 노드 유형 구현이 정의한 구현 아티팩트의 결합으로 구성된다.
  노드 유형 구현이 정의하는 구현 아티팩트는 파생한 노드 유형 구현의 인터페이스와 동작의 이름이 같은 구현 아티팩트에 우선한다.
  노드 유형 구현에서 정의한 구현 아티팩트가 인터페이스 이름만 명시하는 경우, 이는 파생한 노드 유형 구현의 인터페이스 이름이 같은 구현 아티팩트(동작 이름의 정의 여부와는 상관없이)를 대체한다.
  노드 유형 구현에서 정의한 구현 아티팩트가 인터페이스 이름이나 동작 이름을 명시하지 아니하는 경우, 이는 파생한 노드 유형 구현의 모든 구현 아티팩트에 우선한다. 이 경우, 노드 유형의 전체 구현에 우선한다.

- 배포 아티팩트: 노드 유형 구현의 일련의 배포 아티팩트는 노드 유형 구현 자체가 정의하는 배포 아티팩트와 파생한 노드 유형 구현이 정의한 배포 아티팩트의 결합으로 구성된다. 노드 유형 구현이 정의하는 배포 아티팩트는 파생한 노드 유형 구현의 이름과 유형(또는 파생한 유형)의 이름이 같은 배포 아티팩트에 우선한다.

11.4 예시

다음 예제는 노드 유형 구현 “MyDBMSImplementation”을 정의한다. 이는 노드 유형 “DBMS”를 구현한다.

```
<Definitions id="MyImpls" name="My Implementations"
  targetNamespace="http://www.example.com/SampleImplementations"
  xmlns:bn="http://www.example.com/BaseNodeTypes"
  xmlns:ba="http://www.example.com/BaseArtifactTypes"
  xmlns:sa="http://www.example.com/SampleArtifacts">

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

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

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

  <NodeTypeImplementation name="MyDBMSImplementation"
                      nodeType="bn:DBMS">

    <ImplementationArtifacts>
      <ImplementationArtifact interfaceName="MgmtInterface"
```



```
                        artifactType="ba:WARFile"
                        artifactRef="sa:MyMgmtWebApp">
      </ImplementationArtifact>
    </ImplementationArtifacts>

    <DeploymentArtifacts>
      <DeploymentArtifact name="MyDBMS"
                      artifactType="ba:ZipFile"
                      artifactRef="sa:MyInstallable">
      </DeploymentArtifact>
    </DeploymentArtifacts>

  </NodeTypeImplementation>

</Definitions>
```

노드 유형 구현은 “MyDBMSManagement” 구현 아티팩트를 포함한다. 이는 “DBMS” 기본 노드 유형에 대하여 정의한 “MgmtInterface” 인터페이스에 대한 아티팩트이다. 이 아티팩트의 유형은 기본 아티팩트 유형으로 정의한 “WARFile”이다. 구현 아티팩트는 기존에 정의한 “MyMgmtWebApp” 아티팩트 템플릿을 참조한다.

또한 노드 유형 구현은 “MyDBMS” 배포 아티팩트를 포함한다. 이는 “DBMS” 노드 유형의 인스턴스를 만들기 위한 소프트웨어 설치 코드이다. 이 소프트웨어 설치 코드는 “MyInstallable” 아티팩트 템플릿으로써 기존에 별도로 정의한 “ZipFile”이다.

12 관계 유형

이 절은 관계 유형을 정의하는 방법을 명시한다. 관계 유형은 다시 사용할 수 있는 엔티티(노드 템플릿 사이에 하나 이상의 관계 템플릿 유형을 정의하는)이다. 따라서 관계 유형은 관찰 가능한 특성의 구조를 특성 정의(즉 특성이 관계 유형이나 관계 템플릿의 인스턴스를 사용하여 관계 템플릿에서 정의하는 이름, 데이터 유형 및 허용 값)를 통하여 정의할 수 있다.

해당 관계 템플릿에서 인스턴스를 수행할 수 있는 작업은 관계 유형의 인터페이스가 정의한다. 또한 관계 유형은 관계 유형의 인스턴스가 실행 시에 나타내는 잠재적인 상태를 정의한다.

관계 유형은 DerivedFrom 요소를 통하여 다른 관계 유형에서 정의한 바를 상속할 수 있다. 관계 유형을 추상적인 것으로 선언할 수 있다. 이는 인스턴스를 만들 수 없다는 것을 의미한다. 추상적인 관계 유형의 목적은 파생된 전문적인 관계 유형에서 다시 사용하기 위하여 공통의 특성과 행동을 규정하는 것이다. 또한 관계 유형을 최종적인 것으로 선언할 수 있다. 이는 다른 관계 유형이 이를 파생할 수 없다는 것을 의미한다.

12.1 XML 구문

다음 의사 스키마는 관계 유형의 XML 문법을 정의한다.

```
<RelationshipType name="xs:NCName"
                  targetNamespace="xs:anyURI"?
                  abstract="yes|no"?
                  final="yes|no"?>+

  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

  <DerivedFrom typeRef="xs:QName"/> ?

  <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

  <InstanceStates>
    <InstanceState state="xs:anyURI"> +
  </InstanceStates> ?

  <SourceInterfaces>
    <Interface name="xs:NCName| xs:anyURI">
      ...
    </Interface> +
  </SourceInterfaces> ?

  <TargetInterfaces>
    <Interface name="xs:NCName| xs:anyURI">
      ...
    </Interface> +
  </TargetInterfaces> ?

  <ValidSource typeRef="xs:QName"/> ?

  <ValidTarget typeRef="xs:QName"/> ?

</RelationshipType>
```

12.2 속성

RelationshipType 요소는 다음 특성을 가진다.

- name: 이 속성은 관계 유형의 이름이나 식별자를 명시한다. 관계 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- targetNamespace: 이 선택적인(OPTIONAL) 속성은 관계 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 관계 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다.

- abstract: 이 선택적인(OPTIONAL) 속성은 관계 유형을 유형으로써 사용하는 관계

템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다.

결과적으로 늦어도 관계 템플릿을 인스턴스로 만드는 동안 관계 템플릿이 참조하는 추상적인 관계 유형을 추상적인 관계 유형에서 파생된 관계 유형으로 대체하여야 한다.

비고: 추상적인 관계 유형을 최종적인 것으로 선언해서는 안 된다.

- final: 이 선택적인(OPTIONAL) 속성은 이 관계 유형에서 다른 관계 유형을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

비고: 최종 관계 유형을 추상적인 것으로 선언해서는 안 된다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(관계 유형을 작성하는 자가 관계 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  - name: 이 속성은 태그의 이름을 명시한다.

  - value: 이 속성은 태그의 값을 명시한다.
    비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- DerivedFrom: 이는 이 관계 유형이 파생하는 다른 관계 유형을 참조하는 선택적인 요소이다. 로컬 정의가 파생된 정의에 항상 우선한다는 규칙에 따라 서로 모순되는 정의를 해결한다. 자세한 사항에 대해서는 제11.3절 파생 규칙을 참조한다. DerivedFrom 요소는 다음 특성을 가진다.

  - typeRef: QName은 이 관계 유형의 정의가 파생된 출처 관계 유형을 명시한다.

    비고: ValidSource가 노드 유형을 명시하는 경우, 관계 유형의 ValidTarget 요소도 노드 유형을 명시하여야 한다.

    ValidSource가 요구사항 유형을 명시하는 경우, 관계 유형의 ValidTarget 요소는 기능 유형을 명시하여야 한다. 이 기능 유형은 요구사항과 ValidSource에서 정의한 요구사항과 매칭 하여야 한다. 즉 이 기능 유형은 각각의 RequirementType 정의의 requiredCapabilityType 속성에서 명시한 기능의 유형(또는 서브 유형)이어야 한다.

- PropertiesDefinition: 이 요소는 XML 스키마를 통하여 관계 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. PropertiesDefinition 요소는 다음 특성 중 하나의 특성만을 가진다.

  ○ element: 이 속성은 관계 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.

  ○ type: 이 속성은 관계 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.

- InstanceStates: 이 선택적인(OPTIONAL) 요소는 이 관계 유형 인스턴스에 대한 일련의 실행 시 상태를 나열한다. 내포된 InstanceState 요소에서 이런 상태를 정의한다. InstanceState 요소는 다음과 같은 내포된 특성을 가진다.

  ○ state: 이 속성은 잠재적인 상태를 확인하는 URI를 명시한다

- SourceInterfaces: 이 선택적인(OPTIONAL) 요소는 관리 가능한 인터페이스(배포된 서비스에서 출처와 목표 사이의 관계를 실제로 정하기 위하여 이 관계 유형의 관계 출처에서 시행할 수 있는)의 정의를 포함한다.

  내포된 Interface 요소가 이런 인터페이스 정의를 포함한다. 그 내용은 노드 유형 인터페이스에 대하여 서술한 바와 같다(제10.2절 참조).

- TargetInterfaces: 이 선택적인(OPTIONAL) 요소는 관리 가능한 인터페이스(배포된 서비스에서 출처와 목표 사이의 관계를 실제로 정하기 위하여 이 관계 유형의 관계 목표에서 시행할 수 있는)의 정의를 포함한다.
  내포된 Interface 요소가 이런 인터페이스 정의를 포함한다. 그 내용은 노드 유형 인터페이스에 대하여 서술한 바와 같다(제10.2절 참조).

- ValidSource: 이 선택적인(OPTIONAL) 요소는 관계 유형을 사용하여 정의한 관계에 대한 유효한 출처로써 허용되는 객체의 유형을 명시한다. 이를 명시하지 아니하는 경우, 노드 유형이 관계의 출처가 될 수 있다. ValidSource 요소는 다음 특성을 가진다.

  ○ typeRef: 이 속성은 관계 유형을 사용하여 정의하는 관계에 대한 유효한 출처로써 허용되는 노드 유형이나 관계 유형의 QName을 명시한다. 명시한 노드 유형이나 관계 유형으로부터 파생하는 노드 유형이나 관계 유형을 각각 유효한 관계 출처로써 수용하여야 한다(MUST).

  ValidSource가 요구사항 유형을 명시하는 경우, 관계 유형의 ValidTarget 요소는 기능 유형을 명시하여야 한다(MUST). 이 기능 유형은 요구사항과 ValidSource에서 정의한 요구사항과 매칭하여야 한다(MUST). 즉 이 기능 유형은 각각의 RequirementType 정의의 requiredCapabilityType 속성에서 명시한 기능의 유형(또는 서브 유형)이어야 한다(MUST).

- ValidTarget: 이 선택적인(OPTIONAL) 요소는 관계 유형을 사용하여 정의한 관계에 대한 유효한 목표로써 허용되는 객체의 유형을 명시한다. 이를 명시하지 아니하는 경우, 노드 유형이 관계의 출처가 될 수 있다. ValidTarget 요소는 다음 특성을 가진다.

  ○ typeRef: 이 속성은 관계 유형을 사용하여 정의하는 관계에 대한 유효한 목표로써 허용되는 노드 유형이나 기능 유형의 QName을 명시한다. 명시한 노드 유형이나 기능 유형으로부터 파생하는 노드 유형이나 기능 유형을 각각 유효한 관계 목표로써 수용하여야 한다(MUST).

  비고: ValidTarget가 노드 유형을 명시하는 경우, 관계 유형의 ValidSource 요소도 노드 유형을 명시하여야 한다.

  ValidTarget이 기능 유형을 명시하는 경우, 관계 유형의 ValidSource 요소는 요구사항 유형을 명시하여야 한다. 이 요구사항 유형은 ValidTarget에서 정의한 기능을 필요로 한다는 것을 선언하여야 한다. 즉 이 요구사항 유형은 각각의 RequirementType 정의의 requiredCapabilityType 속성에서 명시한 기능의 유형(또는 서브 유형)을 선언하여야 한다.

12.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 관계 유형 특성: 관계 유형의 관계 유형 특성을 의미하는 XML 요소(또는 유형)가 DerivedFrom 요소에서 참조하는 관계 유형의 관계 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

- 인스턴스 상태: 정의 중인 관계 유형의 일련의 인스턴스 상태는 파생된 관계 유형이 정의하는 인스턴스 상태와 정의 중인 관계 유형이 정의하는 인스턴스 상태의 결합으로 구성된다. 이름이 같은 일련의 인스턴스 상태를 결합하여 같은 이름의 하나의 인스턴스 상태로 만든다.

- 유효한 출처 및 목표: 정의 중인 관계 유형의 유효한 출처나 목표라고 각각 명시한

객체의 유형은 파생된 관계 유형의 유효한 출처나 목표라고 각각 정의한 서브 유형과 같아야 한다.

파생된 관계 유형이 유효한 출처나 목표를 정의하지 아니하는 경우, 정의 중인 관계 유형의 ValidSource 또는 ValidTarget에서 정의 중인 객체의 유형을 제한하지 아니한다.
정의 중인 관계 유형이 출처나 목표를 정의하지 아니하는 경우, 파생된 관계 유형의 출처나 목표로써 정의한 객체의 유형만 정의 중인 관계 유형의 유효한 출처나 대상이 된다.

- 인터페이스: 정의 중인 관계 유형의 일련의 인터페이스(출처 및 목표 인터페이스)는 파생된 관계 유형이 정의하는 인터페이스와 정의 중인 관계 유형이 정의하는 인터페이스의 결합으로 구성된다.
  이름이 같은 두 개의 인터페이스를 결합하여 이름이 같은 하나의 파생된 인터페이스로 만든다. 파생된 인터페이스의 여러 동작은 두 인터페이스가 정의하는 일련의 동작의 결합으로 구성된다. 정의 중인 관계 유형이 정의하는 동작은 파생된 관계 유형에서 이름이 같은 동작을 대체한다.

12.4 예시

다음 예제는 관계 유형 “processDeployedOn”을 정의한다. 이 관계 유형은 “프로세스를 호스팅 환경에 배포한다”는 것을 의미한다. 이 관계 유형을 참조하는 관계 템플릿 인스턴스의 소스를 삭제하는 경우, 그 대상도 자동적으로 삭제된다. 관계 유형은 “ProcessDeployedOnProperties” 요소와 같이 같은 정의 문서의 Types 부분에서 정의한 관계 유형 특성을 가진다. 관계 유형 인스턴스가 있을 수 있는 상태도 나열한다.

```
<RelationshipType name="processDeployedOn">

  <RelationshipTypeProperties element="ProcessDeployedOnProperties"/>

  <InstanceStates>
    <InstanceState state="www.example.com/successfullyDeployed"/>
    <InstanceState state="www.example.com/failed"/>
  </InstanceStates>

</RelationshipType>
```


13 관계 유형 구현

이 절은 관계 유형 구현을 정의하는 방법을 명시한다. 관계 유형 구현은 특정한 관계 유형을 구현하는 실행 가능한 코드를 의미한다. 관계 유형 구현은 관계 유형의 인터페이스 동작을 구현하는 실행 가능한 코드 컬렉션(구현 아티팩트라고 알려진)을 규정한다. 특정한 실행 코드를 별도의 아티팩트 템플릿으로써 정의하고 관계 유형 구현의 구현 아티팩트에서 참조한다.

아티팩트 템플릿이 아티팩트에 대한 불변 정보(즉 아티팩트의 파일 이름과 같이 맥락으로부터 독립적인 정보)를 제공하는 반면에 구현 아티팩트는 특정 환경에 대한 인증 데이터와 같은 가변 정보(또는 맥락에 고유한)를 제공할 수 있다.

관계 유형 구현은 TOSCA 컨테이너에 대한 힌트(필수 컨테이너 특징의 정의를 통하여 특정한 환경에 맞는 적절한 구현 방법을 선택할 수 있도록 하는)를 명시할 수 있다.

인터페이스 동작을 정의하지 아니하는(즉 구현 아티팩트를 요구하지 아니하는) 관계 유형이 있을 수 있다는 것에 주의한다. 이런 경우에 관계 유형 구현이 필요하지 아니하며 TOSCA 구현은 각 관계 유형을 사용할 수 있다.

13.1 XML 구문

다음 의사 스키마는 관계 유형 구현의 XML 문법을 정의한다.

```
<RelationshipTypeImplementation name="xs:NCName"
                            targetNamespace="xs:anyURI"?
                            relationshipType="xs:QName"
                            abstract="yes|no"?
                            final="yes|no"?>

  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

  <DerivedFrom relationshipTypeImplementationRef="xs:QName"/> ?

  <RequiredContainerFeatures>
    <RequiredContainerFeature feature="xs:anyURI"/> +
  </RequiredContainerFeatures> ?

  <ImplementationArtifacts>
    <ImplementationArtifact interfaceName="xs:NCName| xs:anyURI"?
                        operationName="xs:NCName"?
                        artifactType="xs:QName"
                        artifactRef="xs:QName"?>
       artifact specific content ?
    <ImplementationArtifact> +
  </ImplementationArtifacts> ?
</RelationshipTypeImplementation>
```

13.2 속성

RelationshipTypeImplementation 요소는 다음 특성을 가진다.

- **name:** 이 속성은 관계 유형 구현의 이름이나 식별자를 명시한다. 관계 유형 구현의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- **targetNamespace:** 이 선택적인(OPTIONAL) 속성은 관계 유형 구현의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 관계 유형 구현을 정의 문서의 목표 네임스페이스에 추가한다.

- **relationshipType:** 이 속성의 QName은 이 관계 유형 구현이 구현하는 관계 유형을 명시하다.

- **abstract:** 이 선택적인(OPTIONAL) 속성은 이 관계 유형 구현을 직접 사용하여 relationshipType 속성에서 명시한 관계 유형을 구현할 수 없다는 것을 명시한다.

  비고: 추상적인 관계 유형 구현을 최종적인 것으로 선언해서는 안 된다.

  예를 들어 관계 유형을 구현하는 자는 다시 사용할 수 있도록 특정 관계 유형의 구현 중 일부만(즉 일부 동작만) 전달하고 특정한 동작을 더 구체적이고 파생된 관계 유형 구현으로 전달하도록 요구할 것을 결정할 수 있다.
- **final:** 이 선택적인(OPTIONAL) 속성은 이 관계 유형 구현에서 다른 관계 유형 구현을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 관계 유형 구현을 추상적인 것으로 선언해서는 안 된다.
- **Tags:** 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(관계 유형 구현을 작성하는 자가 관계 유형 구현을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  ○ **name:** 이 속성은 태그의 이름을 명시한다.

  ○ **value:** 이 속성은 태그의 값을 명시한다.

  비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

  ▪ **DerivedFrom:** 이는 관계 유형 구현이 파생하는 다른 관계 유형 구현을 참조하는 선택적인(OPTIONAL) 요소이다. 자세한 사항에 대해서는 제13.3절 파생 규칙을 참조한다. DerivedFrom 요소는 다음 특성을 가진다.

  ○ **relationshipTypeImplementationRef:** QName은 관계 유형 구현이 파생되는 관계 유형 구현을 명시한다.

- **RequiredContainerFeatures:** TOSCA 컨테이너의 특정한 API(사유 API일 수 있는)와 같이 관계 유형을 실행하는 환경의 특정한 특징에 따라 관계 유형의 구현은 다를 수 있다. 따라서 RequiredContainerFeatures 요소의 콘텐츠는 TOSCA 컨테이너에 "힌트"를 제공하여 복수의 대안이 있는 경우 적절한 관계 유형 구현을 선택할 수 있도록 한다. 별도의 RequiredContainerFeature 요소가 종속성을 정의한다. RequiredContainerFeature 요소는 다음 특성을 가진다.

  ○ **feature:** 이 속성의 값은 환경의 해당 특징을 의미하는 URI이다

- **ImplementationArtifacts:** 이 요소는 관계 유형의 인터페이스나 동작에 대한 일련의 구현 아티팩트를 명시한다. ImplementationArtifacts 요소는 다음 특성을 가진다

  비고: 다음에서 정의하는 속성에 따라 관계 유형을 구현하기 위하여 복수의 구현 아티팩트가 필요할 수도 있다. 구현 아티팩트는 관계 유형에 대하여 정의한 모든 인터페이스와 동작에 대한 구현으로써 역할을 할 수가 있다. 구현 아티팩트는 하나의 인터페이스(및 모든 동작)에 대한 구현으로써 역할을 하거나 하나의 특정한 동작에 대한 구현으로써 역할을 할 수가 있다.

  ○ **ImplementationArtifact:** 이 요소는 인터페이스나 동작에 대하여 하나의 구현 아티팩트를 명시한다.

  ▪ **name:** 이 속성은 아티팩트의 이름을 명시한다. 아티팩트의 이름은 노드 유형 구현의 범위 내에서 고유하여야 한다

  ▪ **interfaceName:** 이 선택적인(OPTIONAL) 속성은 실제 구현 아티팩트가 구현하는 인터페이스의 이름을 명시한다. 명시하지 아니하는 경우, 구현 아티팩트가 RelationshipTypeImplementation의 relationshipType 속성이 참조하는 관계 유형이 정의하는 모든 인터페이스를 구현한다고 가정한다. 이 관계 유형 구현이 구현하는 관계 유형의 SourceInterfaces 요소나 TargetInterfaces 요소에서 참조 인터페이스를 정의할 수 있다는 것에 주의한다.

- operationName: 이 선택적인 속성은 실제 구현 아티팩트가 구현하는 동작의 이름을 명시한다. 명시하는 경우, interfaceName을 명시하고 명시한 operationName은 명시한 인터페이스의 동작을 참조하여야 한다(MUST). 명시하지 아니하는 경우, 구현 아티팩트가 명시한 인터페이스에서 정의한 모든 동작을 구현한다고 가정한다.

- artifactType: 이 속성은 이 아티팩트의 유형을 명시한다. 이 속성의 QName은 같은 정의 문서나 불러온 문서에서 정의한 ArtifactType의 QName과 일치하여야 한다. artifactType 속성은 ImplementationArtifact 요소에 고유한 아티팩트 유형을 명시하고 구현 아티팩트가 artifactRef 속성을 통하여 참조하는 Artifact Template의 유형을 의미한다.

- artifactRef: 이 선택적인(OPTIONAL) 속성은 QName을 포함한다. QName은 구현 아티팩트로써 사용할 아티팩트 템플릿을 확인한다. 같은 정의 문서나 별도의 불러온 문서에서 이 아티팩트 템플릿을 정의할 수 있다. artifactRef 속성이 참조하는 아티팩트 템플릿의 유형은 artifactType 속성에서 명시한 유형과 같은 유형이거나 그 서브 유형이어야 한다(MUST).

  비고: 아티팩트 템플릿을 참조하지 아니하는 경우, ImplementationArtifact 요소에 고유한 아티팩트 유형이 실제 아티팩트를 나타내는 것으로 가정한다. 예를 들어 간단한 스크립트를 ImplementationArtifact 요소에서 정의할 수 있다.

13.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 구현 아티팩트: 관계 유형 구현의 일련의 구현 아티팩트는 관계 유형 구현 자체가 정의하는 구현 아티팩트와 파생된 관계 유형 구현이 정의하는 구현 아티팩트의 조합으로 구성된다.
  노드 유형 구현이 정의하는 구현 아티팩트는 파생된 관계 유형 구현의 인터페이스 이름과 동작 이름이 같은 구현 아티팩트를 우선한다.
  관계 유형 구현에서 정의하는 구현 아티팩트가 인터페이스 이름만 명시하는 경우, 이는 파생된 관계 유형 구현의 인터페이스 이름이 같은(동작 이름의 정의 여부와는 상관없이) 구현 아티팩트를 대체한다. 이 경우, 관계 유형의 인터페이스 구현에 우선한다.
  관계 유형 구현에서 정의한 구현 아티팩트가 인터페이스 이름이나 동작 이름을 정의하지 아니하는 경우, 이는 파생된 관계 유형 구현의 모든 구현 아티팩트에 우선한다. 이 경우, 관계 유형의 전체 구현에 우선한다.

13.4 예시

다음 예제는 노드 유형 구현 “MyDBMSImplementation”을 정의한다. 이는 노드 유형 “DBMS”의 구현이다.

```
<Definitions id="MyImpls" name="My Implementations"
  targetNamespace="http://www.example.com/SampleImplementations"
  xmlns:bn="http://www.example.com/BaseRelationshipTypes"
  xmlns:ba="http://www.example.com/BaseArtifactTypes"
  xmlns:sa="http://www.example.com/SampleArtifacts">

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

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

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

  <RelationshipTypeImplementation name="MyDBConnectImplementation"
                              relationshipType="bn:DBConnection">

    <ImplementationArtifacts>
      <ImplementationArtifact interfaceName="ConnectionInterface"
                          operationName="connectTo"
                          artifactType="ba:ScriptArtifact"
                          artifactRef="sa:MyConnectScript">
      <ImplementationArtifact>
    </ImplementationArtifacts>

  </RelationshipTypeImplementation>

</Definitions>
```

관계 유형 구현은 “MyDBConnectionImpl” 구현 아티팩트를 포함한다. 이는 “DBConnection” 기본 관계 유형에 대하여 정의한 “ConnectionInterface” 인터페이스에 대한 아티팩트이다. 이 아티팩트의 유형은 기본 아티팩트 유형으로 정의한 “ScriptArtifact”이다. 구현 아티팩트는 이전에 정의하였던 “MyConnectScript” 아티팩트 템플릿을 참조한다.

14 요구사항 유형

이 절은 요구사항 유형을 정의하는 방법을 명시한다. 요구사항 유형은 다시 사용할 수 있는 엔티티(노드 유형이 노출시키기 위하여 선언할 수 있는 요구사항의 종류를 서술하는)이다. 예를 들어 데이터베이스 접속에 대한 요구사항 유형을 정의할 수 있으며 데이터베이스 접속에 대한 요구사항을 노출하기(또는 “가지기”) 위하여 다양한 노드 유형(가령 애플리케이션에 대한 노드 유형)을 선언할 수 있다.

요구사항 유형은 특성 정의(즉 노드 유형이 각 요구사항 유형의 요구사항을 정의하는 경우, 노드 템플릿의 요구사항에서 정의한 특성의 이름, 데이터 유형 및 허용 값)를 통하여 가시적인 특성의 구조를 정의한다.

요구사항 유형은 DerivedFrom 요소를 통하여 다른 요구사항 유형으로부터 특성과 의미를 상속할 수 있다. 요구사항 유형을 추상적이라고 선언할 수 있다. 이는 요구사항 유형을 인스턴스로 만들 수 없다는 것을 의미한다. 추상적인 요구사항 유형의 목적은 전문적이고 파생된 요구사항 유형에서 다시 사용하기 위하여 공통의 특성을 규정하는 것이다. 또한 요구사항 유형을 최종적인 것으로 선언할 수도 있다. 이는 다른 요구사항 유형이 파생될 수 없다는 것을 의미한다.

14.1 XML 구문

다음 의사 스키마는 요구사항 유형의 XML 문법을 정의한다.

```
1  <RequirementType name="xs:NCName"
2                targetNamespace="xs:anyURI"?
3                abstract="yes|no"?
4                final="yes|no"?
5                requiredCapabilityType="xs:QName"?>
6
7    <Tags>
8      <Tag name="xs:string" value="xs:string"/> +
9    </Tags> ?
10
11    <DerivedFrom typeRef="xs:QName"/> ?
12
13    <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?
14
15  </RequirementType>
```

14.2 속성

RequirementType 요소는 다음 특성을 가진다.

- name: 이 속성은 요구사항 유형의 이름이나 식별자를 명시한다. 요구사항 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- targetNamespace: 이 선택적인(OPTIONAL) 속성은 요구사항 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 요구사항 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다.

- abstract: 이 선택적인(OPTIONAL) 속성은 이 요구사항 유형을 정의하는 노드 유형의 노드 템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다.

결과적으로 추상적인 요구사항 유형의 요구사항 정의가 있는 노드 유형을 추상적인 것으로 선언하여야 하며 추상적인 요구사항 유형에서 파생되는 유형의 요구사항을 정의하는 파생된 노드 유형도 정의하여야 한다(MUST). 예를 들어 추상적인 유형 "컨테이너"의 요구사항을 가진 것으로 추상적인 노드 유형 "애플리케이션"을 정의할 수 있다. 서비스 템플릿에 따라 서비스를 만드는 동안에 인스턴스화할 수 있는 노드 템플릿을 정의하는 데 사용할 수 있는 "웹 애플리케이션 컨테이너" 유형의 더 구체적인 요구사항을 사용하여 파생된 노드 유형 "웹 애플리케이션"을 정의할 수 있다.

비고: 추상적인 요구사항 유형을 최종적인 것으로 선언해서는 안 된다.

- final: 이 선택적인(OPTIONAL) 속성은 이 요구사항 유형에서 다른 요구사항 유형을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 요구사항 유형을 추상적인 것으로 선언해서는 안 된다.

- requiredCapabilityType: 이 선택적인(OPTIONAL) 속성은 정의한 요구사항 유형을 매칭하기 위하여 필요한 기능의 유형을 명시한다. 이 속성의 QName 값은 같은 정의 문서나 별도의 내포된 문서에서 정의하는 CapabilityType 요소의 QName을 참조한다.

  비고: 각 TOSCA 구현은 요구사항과 기능에 대한 다음과 같은 기본적인 매칭을 지원하여야 한다. 요구사항 정의가 각 요구사항을 정의한다. 이는 요구사항 유형(requiredCapabilityType 속성을 통하여 필요한 요구사항 유형을 명시하는)을 참조한다. 이 속성의 값을 사용하여 유형에 기초한 기본적인 매칭을 한다. 요구사항의 요구사항 유형이 기능이나 그 슈퍼 유형의 기능 유형과 일치하는 requiredCapabilityType 값을 가지는 경우, 기능은 요구사항과 매칭한다.
  해당 요구사항 유형과 기능 유형을 명시하기 위하여 도메인에 고유한 매칭 의미(가령 제약 사항이나 특성에 기초한)를 정의하여야 한다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(요구사항 유형을 작성하는 자가 요구사항 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  - name: 이 속성은 태그의 이름을 명시한다.

  - value: 이 속성은 태그의 값을 명시한다.

비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- **DerivedFrom:** 이는 이 요구사항 유형이 파생하는 다른 요구사항 유형을 참조하는 선택적인 (OPTIONAL)요소이다. 자세한 사항에 대해서는 제14.3절 파생 규칙을 참조한다. **DerivedFrom** 요소는 다음 특성을 가진다.
  - **typeRef:** QName은 이 요구사항 유형의 정의와 의미가 파생된 요구사항 유형을 명시한다.
- **PropertiesDefinition:** 이 요소는 XML 스키마를 통하여 요구사항 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. **PropertiesDefinition** 요소는 다음 특성 중 하나의 특성만을 가진다.
  - **element:** 이 속성은 요구사항 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.
  - **type:** 이 속성은 요구사항 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.

14.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 요구사항 유형 특성: 요구사항 유형 특성을 의미하는 XML 요소(또는 유형)가 **DerivedFrom** 요소에서 참조하는 요구사항 유형의 요구사항 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

14.4 예시

다음 예제는 데이터베이스 접속에 대한 클라이언트의 요구사항을 표현하는 요구사항 유형 "DatabaseClientEndpoint"를 정의한다. 목표 네임스페이스 http://www.example.com/ SampleRequirements에 있는 정의 문서 "MyRequirements"에서 이를 정의한다. 따라서 해당 네임스페이스를 다른 정의 문서로 불러오기를 하여 "DatabaseClientEndpoint" 요구사항 유형을 다른 문서에서 사용할 수 있다.

```
1  <Definitions id="MyRequirements" name="My Requirements"
2    targetNamespace="http://www.example.com/SampleRequirements"
3    xmlns:br="http://www.example.com/BaseRequirementTypes"
4    xmlns:mrp="http://www.example.com/SampleRequirementProperties>
5
6    <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
7      namespace="http://www.example.com/BaseRequirementTypes"/>
```



```
8
9    <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>
```

상기 예제에서 정의한 요구사항 유형 "DatabaseClientEndpoint"는 별도의 파일에서 정의한 다른 일반적인 "ClientEndpoint" 요구사항 유형에서 파생한다(**DerivedFrom** 요소를 통하여). 상기 별도의 Definitions 파일에 있는 정의를 첫 번째 **Import** 요소를 통하여 불러오며 이렇게 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 "**br**"을 할당한다.
"DatabaseClientEndpoint" 요구사항 유형은 XML 스키마 요소 정의 "DatabaseClient EndpointProperties"를 통하여 일련의 특성을 정의한다. 예를 들어 클라이언트 접속을 위하여 사용할 포트 번호의 정의에 이런 특성을 포함시킬 수 있다. XML 스키마 정의를 별도의 XSD 파일(두 번째 **Import** 요소를 통하여 불러오는)에 저장한다. 현재 파일에서 XML 스키마 정의의 네임스페이스에 접두사 "**mrp**"를 할당한다.

15 기능 유형

이 절은 *기능 유형*을 정의하는 방법을 명시한다. 기능 유형은 다시 사용할 수 있는 엔티티(노드 유형이 노출할 것을 선언할 수 있는 기능의 종류를 서술하는)이다. 예를 들어 데이터베이스 서버 엔드 포인트에 대한 기능 유형을 정의할 수 있으며 데이터베이스 서버 엔드 포인트의 역할을 하는 기능을 노출하기(또는 "제공"하기) 위하여 다양한 노드 유형을 선언할 수 있다.

기능 유형은 *특성 정의*(즉 노드 유형이 각 기능 유형의 기능을 정의하는 경우, 노드 유형의 노드 템플릿 중 기능에서 정의하는 특성의 이름, 데이터 유형 및 허용 값)를 통하여 가시적인 특성의 구조를 정의한다.

기능 유형은 **DerivedFrom** 요소를 통하여 다른 기능 유형으로부터 특성과 의미를 상속할 수 있다. 기능 유형을 추상적이라고 선언할 수 있다. 이는 기능 유형을 인스턴스로 만들 수 없다는 것을 의미한다. 추상적인 기능 유형의 목적은 전문적이고 파생된 기능 유형에서 다시 사용하기 위하여 공통의 특성을 규정하는 것이다. 또한 기능 유형을 최종적인 것으로 선언할 수도 있다. 이는 다른 기능 유형이 파생될 수 없다는 것을 의미한다.

15.1 XML 구문

다음 의사 스키마는 기능유형의 XML 문법을 정의한다.

```
<CapabilityType name="xs:NCName"
                targetNamespace="xs:anyURI"?
                abstract="yes|no"?
                final="yes|no"?>

  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

  <DerivedFrom typeRef="xs:QName"/> ?

  <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

</CapabilityType>
```

15.2 속성

`CapabilityType` 요소는 다음 특성을 가진다.

- `name:` 이 속성은 기능 유형의 이름이나 식별자를 명시한다. 기능 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- `targetNamespace:` 이 선택적인(OPTIONAL) 속성은 기능 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 기능 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다.

- `abstract:` 이 선택적인(OPTIONAL) 속성은 이 기능 유형의 기능을 정의하는 노드 유형의 노드 템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다.

  결과적으로 추상적인 기능 유형의 기능 정의가 있는 노드 유형을 추상적인 것으로 선언하여야 하며 추상적인 기능 유형에서 파생되는 유형의 기능을 정의하는 파생된 노드 유형도 정의하여야 한다(MUST). 예를 들어 추상적인 유형 “컨테이너”의 기능을 가진 것으로 추상적인 노드 유형 “서버”를 정의할 수 있다. 서비스 템플릿에 따라 서비스를 만드는 동안에 인스턴스화할 수 있는 노드 템플릿을 정의하는 데 사용할 수 있는 “웹 애플리케이션 컨테이너” 유형의 더 구체적인 기능을 사용하여 파생된 노드 유형 “웹 애플리케이션”을 정의할 수 있다.

  비고: 추상적인 기능 유형을 최종적인 것으로 선언해서는 안 된다.

- `final:` 이 선택적인(OPTIONAL) 속성은 이 기능 유형에서 다른 기능 유형을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 기능 유형을 추상적인 것으로 선언해서는 안 된다.

- `Tags:` 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(기능 유형을 작성하는 자가 기능 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. `Tag` 요소는 다음 특성을 가진다.

  - `name:` 이 속성은 태그의 이름을 명시한다.
  - `value:` 이 속성은 태그의 값을 명시한다.

    비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- `DerivedFrom:` 이는 이 기능 유형이 파생하는 다른 기능 유형을 참조하는 선택적인(OPTIONAL)요소이다. 자세한 사항에 대해서는 제15.3절 파생 규칙을 참조한다. `DerivedFrom` 요소는 다음 특성을 가진다.

  - `typeRef:` QName은 이 기능 유형의 정의와 의미가 파생된 기능 유형을 명시한다.

- `PropertiesDefinition:` 이 요소는 XML 스키마를 통하여 기능 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. `PropertiesDefinition` 요소는 다음 특성 중 하나의 특성만을 가진다.

  - `element:` 이 속성은 기능 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.
  - `type:` 이 속성은 기능 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.

15.3 파생 규칙

`DerivedFrom`에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 기능 유형 특성: 기능 유형 특성을 의미하는 XML 요소(또는 유형)가 `DerivedFrom` 요소에서 참조하는 기능 유형의 기능 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

15.4 예시

다음 예제는 데이터베이스 접속에 대한 클라이언트의 기능을 표현하는 기능 유형 “DatabaseServerEndpoint”를 정의한다. 목표 네임스페이스 http://www.example.com/SampleCapabilities에 있는 정의 문서 “MyCapabilities”에서 이를 정의한다. 따라서 해당 네임스페이스를 다른 정의 문서로 불러오기를 하여 “DatabaseServerEndpoint” 기능 유형을 다른 문서에서 사용할 수 있다.

```
<Definitions id="MyCapabilities" name="My Capabilities"
  targetNamespace="http://www.example.com/SampleCapabilities"
  xmlns:bc="http://www.example.com/BaseCapabilityTypes"
  xmlns:mcp="http://www.example.com/SampleCapabilityProperties>

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

  <Import importType="http://www.w3.org/2001/XMLSchema"
         namespace="http://www.example.com/SampleCapabilityProperties"/>

   <CapabilityType name="DatabaseServerEndpoint">
     <DerivedFrom typeRef="bc:ServerEndpoint"/>
     <PropertiesDefinition
       element="mcp:DatabaseServerEndpointProperties"/>
   </CapabilityType>

</Definitions>
```

상기 예제에서 정의한 기능 유형 “DatabaseServerEndpoint”는 별도의 파일에서 정의한 다른 일반적인 “ServerEndpoint” 기능 유형에서 파생한다(**DerivedFrom** 요소를 통하여). 상기 별도의 Definitions 파일에 있는 정의를 첫 번째 **Import** 요소를 통하여 불러오며 이렇게 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 “**bc**”을 할당한다. “DatabaseServerEndpoint” 기능 유형은 XML 스키마 요소 정의 “DatabaseServerEndpointProperties”를 통하여 일련의 특성을 정의한다. 예를 들어 클라이언트 접속을 듣기 위하여 서버가 사용할 포트 번호나 클라이언트가 사용할 인증의 정의에 이런 특성을 포함시킬 수 있다. XML 스키마 정의를 별도의 XSD 파일(두 번째 **Import** 요소를 통하여 불러오는)에 저장한다. 현재 파일에서 XML 스키마 정의의 네임스페이스에 접두사 “**mcp**”를 할당한다.

16 아티팩트 유형

이 절은 *아티팩트* 유형을 정의하는 방법을 명시한다. 아티팩트 유형은 다시 사용할 수 있는 엔티티(하나 이상의 아티팩트 템플릿의 유형을 정의하는)이다. 이는 노드 템플릿에 대한 배포 아티팩트의 역할이나 노드 유형 및 관계 유형 인터페이스 동작에 대한 구현 아티팩트의 역할을 한다. 예를 들어 웹 애플리케이션 아카이브 파일을 서술하기 위하여 아티팩트 유형 “WAR File”을 정의할 수 있다. 이 아티팩트 유형에 기초하여 하나 이상의 아티팩트 템플릿(구체적인 WAR 파일을 의미하는)을 정의하고 배포 또는 구현 아티팩트

로써 참조할 수 있다.

아티팩트 유형은 *특성 정의*(아티팩트 유형이나 이런 아티팩트 유형의 인스턴스를 사용하여 아티팩트 템플릿에서 정의하는 특성의 이름, 데이터 유형 및 허용 값)를 통하여 가시적인 특성의 구조를 정의할 수 있다. 상황에 따라 변할 수 있은 특성과는 달리 아티팩트 유형이 정의하는 특성이 상황(해당 아티팩트를 사용하는)에 따라 변하지 아니한다고 가정하는 것에 주목한다. 이런 변하지 아니하는 특성의 예로써 WAR 파일에 대한 아티팩트 유형은 실제 아티팩트가 적절한지를 인증하기 위한 해시를 유지할 수 있는 “서명” 특성을 정의할 수 있다. 이와 대조적으로 WAR 파일에 포함된 웹 애플리케이션을 배포하는 경로는 WAR 파일이 사용되는 위치에 따라 다를 수 있다.

아티팩트 유형은 **DerivedFrom** 요소를 통하여 다른 아티팩트 유형으로부터 특성과 의미를 상속할 수 있다. 아티팩트 유형을 추상적이라고 선언할 수 있다. 이는 아티팩트 유형을 인스턴스로 만들 수 없다는 것을 의미한다. 추상적인 아티팩트 유형의 목적은 전문적이고 파생된 아티팩트 유형에서 다시 사용하기 위하여 공통의 특성을 규정하는 것이다. 또한 아티팩트 유형을 최종적인 것으로 선언할 수도 있다. 이는 다른 아티팩트 유형이 파생될 수 없다는 것을 의미한다.

16.1 XML 구문

다음 의사 스키마는 아티팩트 유형의 XML 문법을 정의한다.

```
<ArtifactType name="xs:NCName"
              targetNamespace="xs:anyURI"?
              abstract="yes|no"?
              final="yes|no"?>

  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

   <DerivedFrom typeRef="xs:QName"/> ?

   <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

</ArtifactType>
```

16.2 속성

ArtifactType 요소는 다음 특성을 가진다.

- **name:** 이 속성은 아티팩트 유형의 이름이나 식별자를 명시한다. 아티팩트 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- **targetNamespace:** 이 선택적인(OPTIONAL) 속성은 아티팩트 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 아티팩트 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다.

- **abstract:** 이 선택적인(OPTIONAL) 속성은 이 기능 유형의 기능을 정의하는 아티팩트 유형의 아티팩트 템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다. 즉 어떠한 경우에도 각 아티팩트를 배포 또는 구현 아티팩트로써 직접 사용할 수 없다.

  결과적으로 늦어도 아티팩트(즉 노드 템플릿이나 관계 템플릿)를 사용하는 요소를 배포하는 동안에 추상적인 아티팩트 유형의 아티팩트 템플릿을 파생된 아티팩트 유형의 아티팩트로 대체하여야 한다.

  비고: 추상적인 아티팩트 유형을 최종적인 것으로 선언해서는 안 된다.

- **final:** 이 선택적인 속성은 이 아티팩트 유형에서 다른 아티팩트 유형을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 아티팩트 유형을 추상적인 것으로 선언해서는 안 된다.

- **Tags:** 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(아티팩트 유형을 작성하는 자가 아티팩트 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 **Tag** 요소를 통하여 각 태그를 정의한다. **Tag** 요소는 다음 특성을 가진다.

  - **name:** 이 속성은 태그의 이름을 명시한다.

  - **value:** 이 속성은 태그의 값을 명시한다.

    비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- **DerivedFrom:** 이는 이 아티팩트 유형이 파생하는 다른 아티팩트 유형을 참조하는 선택적인(OPTIONAL) 요소이다. 자세한 사항에 대해서는 제16.3절 파생 규칙을 참조한다. **DerivedFrom** 요소는 다음 특성을 가진다.

  - **typeRef:** QName은 이 아티팩트 유형의 정의와 의미가 파생된 아티팩트 유형을 명시한다.

- **PropertiesDefinition:** 이 요소는 XML 스키마를 통하여 아티팩트 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. **PropertiesDefinition** 요소는 다음 특성 중 하나의 특성만을 가진다.

  - **element:** 이 속성은 아티팩트 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.

  - **type:** 이 속성은 아티팩트 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.

16.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 아티팩트 유형 특성: 아티팩트 유형 특성을 의미하는 XML 요소(또는 유형)가 **DerivedFrom** 요소에서 참조하는 아티팩트 유형의 아티팩트 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

16.4 예시

다음 예제는 다양한 Linux에 대하여 배포할 수 있는 아티팩트로써 RPM 패키지를 서술하는 데 사용할 수 있는 아티팩트 유형 "RPMPackage"를 정의한다. 목표 네임스페이스 http://www.example.com/SampleArtifacts에 있는 정의 문서 "MyArtifacts"에서 이를 정의한다. 따라서 해당 네임스페이스를 다른 정의 문서로 불러오기를 하여 "RPMPackage" 아티팩트 유형을 다른 문서에서 사용할 수 있다.

```
<Definitions id="MyArtifacts" name="My Artifacts"
  targetNamespace="http://www.example.com/SampleArtifacts"
  xmlns:ba="http://www.example.com/BaseArtifactTypes"
  xmlns:map="http://www.example.com/SampleArtifactProperties>

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

  <Import importType="http://www.w3.org/2001/XMLSchema"
         namespace="http://www.example.com/SampleArtifactProperties"/>

   <ArtifactType name="RPMPackage">
     <DerivedFrom typeRef="ba:OSPackage"/>
     <PropertiesDefinition element="map:RPMPackageProperties"/>
   </ArtifactType>

</Definitions>
```

상기 예제에서 정의한 아티팩트 유형 "RPMPackage"는 별도의 파일에서 정의한 다른 일

반적인 “OSPackage” 아티팩트 유형에서 파생한다(DerivedFrom 요소를 통하여). 상기 별도의 Definitions 파일에 있는 정의를 첫 번째 Import 요소를 통하여 불러오며 이렇게 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 “ba”을 할당한다.
“RPMPackage” 아티팩트 유형은 XML 스키마 요소 정의 “RPMPackageProperties”를 통하여 일련의 특성을 정의한다. 예를 들어 상기 특성은 하나 이상의 RPM 패키지의 정의를 포함할 수 있다. XML 스키마 정의를 별도의 XSD 파일(두 번째 Import 요소를 통하여 불러오는)에 저장한다. 현재 파일에서 XML 스키마 정의의 네임스페이스에 접두사 “map”를 할당한다.

17 아티팩트 템플릿

이 절은 *아티팩트* 템플릿을 정의하는 방법을 명시한다. 아티팩트 템플릿은 서비스 템플릿에 있는 다른 객체에서 배포 아티팩트나 구현 아티팩트로써 참조할 수 있는 아티팩트를 의미한다. 예를 들어 노드 유형이나 노드 템플릿에서 설치 가능한 소프트웨어를 위한 아티팩트 템플릿을 특정한 소프트웨어 컴포넌트를 구체화하기 위한 배포 아티팩트로써 참조할 수 있다. 또 다른 예제로써 노드 유형이나 관계 유형의 인터페이스 정의에서 WAR 파일에 대한 아티팩트 템플릿을 REST 동작에 대한 구현 아티팩트로써 참조할 수 있다.

아티팩트 템플릿은 가시적인 특성(메타 데이터)이나 아티팩트의 구조를 정의하는 특정한 아티팩트 유형을 참조한다. 일반적으로 아티팩트 템플릿은 Properties 요소에 있는 특성에 대한 값을 정의한다. 상황에 따라 변할 수 있는 특성과는 달리 아티팩트 유형이 정의하는 특성이 상황(해당 아티팩트를 사용하는)에 따라 변하지 아니한다고 가정하는 것에 주의한다.

또한 일반적으로 아티팩트 템플릿은 전체적인 서비스 템플릿을 포함하는 CSAR에 파일로써 포함되거나 FTP 서버와 같은 원격 위치에서 사용할 수 있는 실제 아티팩트 자체에 대한 하나 이상의 참조를 규정한다.

17.1 XML 구문

다음 의사 스키마는 아티팩트 템플릿의 XML 문법을 정의한다.

```
<ArtifactTemplate id="xs:ID" name="xs:string"? type="xs:QName">

  <Properties>
    XML fragment
  </Properties> ?

  <PropertyConstraints>
    <PropertyConstraint property="xs:string"
                        constraintType="xs:anyURI"> +
      constraint ?
    </PropertyConstraint>
  </PropertyConstraints> ?

  <ArifactReferences>
    <ArtifactReference reference="xs:anyURI">
      (
       <Include pattern="xs:string"/>
      |
       <Exclude pattern="xs:string"/>
      )*
    </ArtifactReference> +
  </ArtifactReferences> ?

</ArtifactTemplate>
```

17.2 속성

ArtifactTemplate 요소는 다음 특성을 가진다.

- id: 이 속성은 아티팩트 템플릿의 식별자를 명시한다. 아티팩트 템플릿의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- name: 이 선택적인(OPTIONAL) 속성은 아티팩트 템플릿의 이름을 명시한다.

- type: 이 속성의 QName 값은 아티팩트 템플릿의 유형을 규정하는 아티팩트 유형을 참조한다.

  비고: 아티팩트 템플릿의 type 속성이 참조하는 아티팩트 유형을 추상적이라고 선언하는 경우, 특정한 아티팩트 템플릿의 인스턴스를 만들 수 없다. 즉 아티팩트를 배치 또는 구현 아티팩트로써 직접 사용할 수 없다. 대신에 늦어도 서비스 템플릿의 인스턴스를 만드는 동안 전문적이고 파생된 아티팩트 유형이 있는 관계 템플릿으로 아티팩트 템플릿을 대체하여야 한다.

- Properties: 이 선택적인(OPTIONAL) 요소는 아티팩트 템플릿의 불변 특성(즉 아티팩트 템플릿을 사용하는 여러 다른 상황에서 흔히 사용되는 특성)을 명시한다.

  해당 아티팩트 유형 특성의 XML 스키마의 인스턴스 문서를 규정함으로써 초기값을 명시한다. 이 인스턴스 문서는 아티팩트 템플릿의 type 속성이 참조하는 아티팩트 유형의 DerivedFrom 특성에서 비롯하는 상속 구조를 고려한다.

- PropertyConstraints: 이 선택적인(OPTIONAL) 요소는 아티팩트 템플릿의 특성을 정의하는 아티팩트 유형에 대하여 하나 이상의 아티팩트 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 PropertyConstraint 요소를

통하여 각 제약 사항을 명시한다. PropertyConstraint 요소는 다음 특성을 가진다.

- property: 이 특성의 문자열 값은 아티팩트 유형 특성 문서(아티팩트 템플릿의 맥락에서 제한된)에서 특성을 가리키는 XPath 표현식이다. 각 특성에 대하여 하나 이상의 제약 사항을 정의하지 아니하여야 한다(MUST NOT).

- constraintType: URI를(제약 사항의 의미와 콘텐츠의 포맷을 정의하는) 통하여 제약 사항 유형을 명시한다.

  예를 들어 http://www.example.com/PropertyConstraints/unique의 제약 사항 유형은 정의에 따른 아티팩트 템플릿의 참조 특성이 특정 범위 내에서 고유하여야 한다는 것을 의미할 수 있다. 각 PropertyConstraint 요소의 제약 사항 유형은 고유함을 보장하여야 하는 실제 범위를 더 상세하게 정의할 수 있다

- ArtifactReferences: 이 선택적인(OPTIONAL) 요소는 실제 아티팩트에 대한 하나 이상의 참조를(별도의 ArtifactReference 요소가 표현하는) 포함한다. ArtifactReference 요소는 다음 특성을 가진다.

  - reference: 이 URI가 상대 URI인 경우, 이를 서비스 템플릿을 포함하는 CSAR의 루트 디렉터리에 대하여 상대적으로 해석한다(제7.7절과 제18절을 참조한다).

  - Include: 참조가 전체 디렉터리를 가리키는 경우, 이 선택적인(OPTIONAL) 요소를 사용하여 아티팩트 참조에 포함시킬 파일의 패턴을 정의할 수 있다. Include 요소는 다음 특성을 가진다.

    - pattern: 이 속성은 전체 아티팩트 참조에 포함하여야 하는 파일에 대한 패턴 정의를 포함한다. 예를 들어 "*.py"의 패턴은 디렉터리에 포함된 모든 python 스크립트를 포함할 것이다.

  - Exclude: 참조가 전체 디렉터리를 가리키는 경우, 이 선택적인(OPTIONAL) 요소를 사용하여 아티팩트 참조에서 제외할 파일의 패턴을 정의할 수 있다. Exclude 요소는 다음 특성을 가진다.

    - pattern: 이 속성은 전체 아티팩트 참조에서 제외하여야 하는 파일에 대한 패턴 정의를 포함한다. 예를 들어 "*.sh"의 패턴은 디렉터리에 포함된 모든 bash 스크립트를 제외할 것이다.

17.3 예시

다음 예제는 일부 소프트웨어 설치 코드를 포함하는 zip 파일을 가리키는 아티팩트 템플릿 "MyInstallable"을 정의한다. 목표 네임스페이스 "http://www.example.com/SampleArtifacts"에 있는 정의 문서 "MyArtifacts"에서 이를 정의한다. 예를 들어 소프트웨어 컴포넌트를 의미하는 노드 템플릿에 대한 배포 아티팩트로써 같은 문서에서 아티팩트 템플릿을 사용하거나 해당 네임스페이스를 다른 문서로 불러와서 다른 문서에서 사용할 수 있다.

```
<Definitions id="MyArtifacts" name="My Artifacts"
  targetNamespace="http://www.example.com/SampleArtifacts"
  xmlns:ba="http://www.example.com/BaseArtifactTypes">

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

  <ArtifactTemplate id="MyInstallable"
                name="My installable"
                 type="ba:ZipFile">
    <ArtifactReferences>
      <ArtifactReference reference="files/MyInstallable.zip"/>
    </ArtifactReferences>
  </ArtifactTemplate>

</Definitions>
```

상기 예제에서 정의한 아티팩트 유형 "MyInstallable"은 ArtifactTemplate 요소의 type 속성에서 명시하는 "ZipFile" 유형이다 이 아티팩트 유형을 별도의 파일에서 정의한다. 상기 별도의 파일에 있는 정의를 Import 요소를 통하여 불러오며 이렇게 불러온 정의에 대하여 현재 파일에서 접두사 "ba"을 할당한다.

"MyInstallable" 아티팩트 유형은 ArtifactReference 요소를 통하여 "MyInstallable.zip" 파일에 대한 참조를 규정한다. reference 속성에서 규정한 URI는 상대 URI이기 때문에 이를 서비스 템플릿을 포함하는 CSAR의 루트 디렉터리에 대하여 상대적으로 해석한다.

18 정책 유형

이 절은 정책 유형을 정의하는 방법을 명시한다. 정책 유형은 다시 사용할 수 있는 엔티티(노드 유형이 노출할 것을 선언할 수 있는 비기능적인 행동의 종류나 서비스 품질을 서술하는)이다. 예를 들어 특정 노드 유형(가령 애플리케이션 서버를 위한 노드 유형)에 대하여 높은 사용 가능성을 표현하기 위하여 정책 유형을 정의할 수 있다.
정책 유형은 특성 정의(해당 정책 템플릿에서 정의하는 특성의 이름, 데이터 유형 및 허용 값)를 통하여 가시적인 특성의 구조를 정의한다.
정책 유형은 DerivedFrom 요소를 통하여 다른 정책 유형으로부터 특성을 상속할 수 있다.

정책 유형은 일련의 노드 유형(AppliesTo 요소를 통하여 비기능적 행동을 명시하는)을 선언한다. “적용할 수 있는” 것이 구현하지는 아니한다는 것에 주의한다. 즉 높은 사용 가능성을 표현하는 정책 유형이 “Webserver” 노드 유형과 관련되는 경우, 웹 서버의 인스턴스의 사용 가능성이 반드시 높은 것은 아니다. 정책 유형을 적용할 수 있는 노드 유형의 인스턴스가 명시한 비기능적인 행동을 보여줄 것인지는 해당 노드 유형의 노드 템플릿이 결정한다.

18.1 XML 구문

다음 의사 스키마는 정책 유형의 XML 문법을 정의한다.

```
<PolicyType name="xs:NCName"
          policyLanguage="xs:anyURI"?
          abstract="yes|no"?
          final="yes|no"?
          targetNamespace="xs:anyURI"?>
  <Tags>
    <Tag name="xs:string" value="xs:string"/> +
  </Tags> ?

  <DerivedFrom typeRef="xs:QName"/> ?

  <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

  <AppliesTo>
    <NodeTypeReference typeRef="xs:QName"/> +
  </AppliesTo> ?

  policy type specific content ?

</PolicyType>
```

18.2 속성

PolicyType 요소는 다음 특성을 가진다.

- name: 이 속성은 정책 유형의 이름이나 식별자를 명시한다. 정책 유형의 이름이나 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- targetNamespace: 이 선택적인(OPTIONAL) 속성은 정책 유형의 정의를 추가하는 목표 네임스페이스를 명시한다. 명시하지 아니하는 경우, 정책 유형 정의를 정의 문서의 목표 네임스페이스에 추가한다

- policyLanguage: 이 선택적인(OPTIONAL) 속성은 정책 유형의 세부 사항을 명시하는 데 사용하는 언어를 명시한다(MUST). 이런 세부 사항을 정책 유형에 고유한 정책 유형 요소의 콘텐츠로써 정의할 수 있다.

- abstract: 이 선택적인(OPTIONAL) 속성은 추상적인 정책 유형의 정책 템플릿에서 인스턴스를 만들 수 없다는 것을 명시한다. 즉 서비스 템플릿의 인스턴스를 만드는 동안에는 각각의 정책을 사용할 수 없다.

  결과적으로 늦어도 정책이 첨부된 요소를 배포하는 동안 추상적인 정책 유형의 정책 템플릿을 파생 정책 유형의 정책으로 교체하여야 한다.

- final: 이 선택적인(OPTIONAL) 속성은 이 정책 유형에서 다른 정책 유형을 파생하여서는 아니 된다는 것을 명시한다(MUST NOT).

  비고: 최종 정책 유형을 추상적인 것으로 선언해서는 안 된다.

- Tags: 이 선택적인(OPTIONAL) 요소를 통하여 많은 태그(정책 유형을 작성하는 자가 정책 유형을 서술하는 데 사용할 수 있는)를 정의할 수 있다. 별도의 내포된 Tag 요소를 통하여 각 태그를 정의한다. Tag 요소는 다음 특성을 가진다.

  - name: 이 속성은 태그의 이름을 명시한다.
  - value: 이 속성은 태그의 값을 명시한다.

  비고: 태그에서 정의한 이름/값의 쌍을 규범적으로 해석하지 않는다.

- DerivedFrom: 이는 이 정책 유형이 파생하는 다른 정책 유형을 참조하는 선택적인 요소이다. 자세한 사항에 대해서는 제18.3절 파생 규칙을 참조한다.
  DerivedFrom 요소는 다음 특성을 가진다.

  - typeRef: QName은 이 정책 유형의 정의와 의미가 파생된 정책 유형을 명시한다.

- PropertiesDefinition: 이 요소는 XML 스키마를 통하여 정책 유형 특성의 구조(구성 및 상태와 같은)를 명시한다. PropertiesDefinition 요소는 다음 특성 중 하나의 특성만을 가진다.

  - element: 이 속성은 정책 유형 특성의 구조를 정의하는 XML 요소의 QName을 규정한다.

  - type: 이 속성은 정책 유형 특성의 구조를 정의하는 XML (복잡한) 유형의 QName을 규정한다.

- `AppliesTo:` 이 선택적인(OPTIONAL) 요소는 정책 유형을 적용할 수 있는 일련의 노드 유형을 명시한다. 각 노드 유형을 별도의 내포된 `NodeTypeReference` 요소로써 정의한다. `NodeTypeReference` 요소는 다음 특성을 가진다.

    - `typeRef:` 이 속성은 정책 유형을 적용하는 노드 유형의 QName을 규정한다.

18.3 파생 규칙

DerivedFrom에 기초하여 정의를 결합하는 데 다음 규칙을 적용한다.

- 특성 정의: 정책 유형 특성을 의미하는 XML 요소(또는 유형)가 `DerivedFrom` 요소에서 참조하는 정책 유형의 정책 유형 특성의 XML 요소(또는 특성)를 확장한다고 가정한다.

- 적용: 정책 유형을 적용할 수 있는 일련의 노드 유형은 정책 유형이 `AppliesTo` 요소를 통하여 명시적으로 참조하는 노드 유형과 파생한 노드 유형의 결합으로 구성된다.
- 정책 언어: 정책 유형은 자신이 파생된 정책 유형의 정책 언어와 같은 정책 언어를 정의하여야 한다. 파생의 근거로써 사용된 정책 유형이 `policyLanguage` 속성을 정의하지 아니하는 경우, 파생된 정책 유형은 적절한 정책 언어를 정의할 수 있다.

18.4 예시

다음 예제는 두 가지의 정책 유형인 “HighAvailability” 정책 유형과 “ContinuousAvailability” 정책 유형을 정의한다. 목표 네임스페이스 http://www.example.com/SamplePolicyTypes에 있는 정의 문서 “MyPolicyTypes”에서 이를 정의한다. 따라서 해당 네임스페이스를 다른 정의 문서로 불러오기를 하여 두 가지의 정책 유형을 다른 문서에서 사용할 수 있다.

```
<Definitions id="MyPolicyTypes" name="My Policy Types"
  targetNamespace="http://www.example.com/SamplePolicyTypes"
  xmlns:bnt="http://www.example.com/BaseNodeTypes">
  xmlns:spp="http://www.example.com/SamplePolicyProperties">

  <Import importType="http://www.w3.org/2001/XMLSchema"
        namespace="http://www.example.com/SamplePolicyProperties"/>

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


   <PolicyType name="HighAvailability">
     <PropertiesDefinition element="spp:HAProperties"/>
   </PolicyType>

   <PolicyType name="ContinuousAvailability">
     <DerivedFrom typeRef="HighAvailability"/>
     <PropertiesDefinition element="spp:CAProperties"/>
     <AppliesTo>
       <NodeTypeReference typeRef="bnt:DBMS"/>
     </AppliesTo>
   </PolicyType>

 </Definitions>
```


상기 예제에서 정의한 정책 유형 “HighAvailability”는 별도의 네임스페이스에서 XML 요소로써 정의한 “HAProperties” 특성을 가진다. 같은 네임스페이스는 “ContinuousAvailability” 정책 유형의 특성을 정의하는 “CAProperties” 요소를 포함한다. 첫 번째 `Import` 요소를 통하여 불러오며 이렇게 불러온 네임스페이스와 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 “spp”를 할당한다.

“HighAvailability” 정책 유형에서 “ContinuousAvailability” 정책 유형이 파생한다. 또한 이를 “DBMS” 노드 유형에 적용할 수 있다. 별도의 네임스페이스에서 이 노드 유형을 정의하고 두 번째 `Import` 요소를 통하여 이 노드 유형을 불러온다. 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 “bnt”를 할당한다.

19 정책 템플릿

이 절은 *정책 템플릿*을 정의하는 방법을 명시한다. 정책 템플릿은 노드 템플릿이 참조할 수 있는 특정한 비기능적 행동이나 서비스 품질을 의미한다. 정책 템플릿은 비기능적 행동의 가시적인 특성(메타 데이터)의 구조를 정의하는 특정한 정책 유형을 참조한다. 일반적으로 아티팩트 템플릿은 `Properties` 요소에 있는 특성에 대한 값을 정의한다. 상황에 따라 변할 수 있는 노드 템플릿의 정책에서 정의한 특성과는 달리 정책 템플릿이 정의하는 특성이 상황(해당 행동을 노출하는)에 따라 변하지 아니한다고 가정하는 것에 주의한다.

19.1 XML 구문

다음 의사 스키마는 정책 템플릿의 XML 문법을 정의한다.

```
<PolicyTemplate id="xs:ID" name="xs:string"? type="xs:QName">

  <Properties>
    XML fragment
  </Properties> ?

  <PropertyConstraints>
    <PropertyConstraint property="xs:string"
                        constraintType="xs:anyURI"> +
      constraint ?
    </PropertyConstraint>
  </PropertyConstraints> ?

```

```
14    policy type specific content ?
15
16  </PolicyTemplate>
```

19.2 속성

PolicyTemplate 요소는 다음 특성을 가진다.

- **id:** 이 속성은 정책 템플릿의 식별자를 명시한다. 정책 템플릿의 식별자는 목표 네임스페이스에서 고유하여야 한다(MUST).

- **name:** 이 선택적인(OPTIONAL) 속성은 정책 템플릿의 이름을 명시한다.

- **type:** 이 속성의 QName 값은 정책 템플릿의 유형을 규정하는 정책 유형을 참조한다.

- **Properties:** 이 선택적인(OPTIONAL) 요소는 정책 템플릿의 불변 특성(즉 정책 템플릿을 사용하는 여러 다른 상황에서 흔히 사용되는 특성)을 명시한다.

  해당 정책 유형 특성의 XML 스키마의 인스턴스 문서를 규정함으로써 초기값을 명시한다. 이 인스턴스 문서는 정책 템플릿의 **type** 속성이 참조하는 정책 유형의 **DerivedFrom** 특성에서 비롯하는 상속 구조를 고려한다.

- **PropertyConstraints:** 이 선택적인(OPTIONAL) 요소는 정책 템플릿의 특성을 정의하는 정책 유형에 대하여 하나 이상의 정책 유형 특성을 사용하는 것에 대한 제약 사항을 명시한다. 별도의 내포된 **PropertyConstraint** 요소를 통하여 각 제약 사항을 명시한다. **PropertyConstraint** 요소는 다음 특성을 가진다.

  - **property:** 이 특성의 문자열 값은 정책 유형 특성 문서(정책 템플릿의 맥락에서 제한된)에서 특성을 가리키는 XPath 표현식이다. 각 특성에 대하여 하나 이상의 제약 사항을 정의하지 아니하여야 한다(MUST NOT).

  - **constraintType:** URI를(제약 사항의 의미와 콘텐츠의 포맷을 정의하는) 통하여 제약 사항 유형을 명시한다.

19.3 예시

다음 예제는 정책 템플릿 "MyHAPolicy"를 정의한다. 이를 목표 네임스페이스 http://www.example.com/SamplePolicies에 있는 정의 문서 "MyPolicies"에서 정의한다. 정책 템플릿을 같은 정의 문서에서(예를 들어 일부 노드 템플릿의 정책으로써) 사용하거나 해당 네임스페이스를 다른 문서로 불러와서 다른 문서에서 사용할 수 있다.

```
1  <Definitions id="MyPolices" name="My Policies"
2    targetNamespace="http://www.example.com/SamplePolicies"
3    xmlns:spt="http://www.example.com/SamplePolicyTypes">
4
5    <Import importType="http://docs.oasis-open.org/tosca/ns/2011/12"
6          namespace="http://www.example.com/SamplePolicyTypes"/>
7
8    <PolicyTemplate id="MyHAPolicy"
9                 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>
```

상기 예제에서 정의한 정책 유형 "MyHAPolicy"는 **PolicyTemplate** 요소의 **type** 속성에서 명시하는 "HighAvailability" 유형이다. 이 정책 유형을 별도의 파일에서 정의한다. 상기 별도의 파일에 있는 정의를 **Import** 요소를 통하여 불러오며 이렇게 불러온 정의와 불러온 정의의 네임스페이스에 대하여 현재 파일에서 접두사 "spt"을 할당한다.
"MyHAPolicy" 정책 유형은 "HighAvailability" 정책 유형의 특성 정의가 정의하는 특성에 대한 값을 규정한다. **AvailabilityClass** 특성의 값을 4로 한다. **HeartbeatFrequency**의 값을 "250"으로 한다.

20 클라우드 서비스 아카이브(CSAR)

이 절은 클라우드 서비스 아카이브의 메타데이터와 전체적인 구조를 정의한다.

20.1 CSAR의 전체 구조

CSAR은 최소한 두 개의 디렉터리(**TOSCA-Metadata**와 **Definitions**)를 포함하는 zip 파일이다. CSAR는 언급된 두 디렉터리 외에 다른 디렉터리도 포함할 수 있다(MAY). CSAR를 만들 때, 클라우드 애플리케이션에 적합하도록 CSAR의 콘텐츠를 자유로이 정의하고 구조화할 수가 있다.

TOSCA-Metadata 디렉터리는 CSAR의 다른 콘텐츠를 서술하는 메타데이터를 포함하는데 이 메타데이터를 *TOSCA 메타파일*이라고 부르고 **TOSCA**라는 파일명으로 확장자로 **.meta**가 붙는다.

Definitions 디렉터리는 하나 이상의 TOSCA 정의서(파일 확장자 .tosca)를 포함한다. 일반적으로 이런 정의서 파일은 해당 CSAR에 대한 클라우드 애플리케이션과 관련된 정의들을 포함한다. 또한 CSAR은 다른 곳에서 재사용하기 위한 요소들의 정의만 포함할 수 있다. 예를 들어 CSAR을 사용하여 일련의 노드 유형 및 관계 유형을 구현체와 함께 패키지로 만들 수 있는데 이런 경우 다른 CSAR에 있는 서비스 템플릿이 해당 패키지를 사용할 수가 있다. 클라우드 애플리케이션 전체를 CSAR에 패키지로 만드는 경우, Definitions 디렉터리에 있는 정의서 중 하나는 해당 클라우드 애플리케이션의 구조와 행위를 정의하는 서비스 템플릿 정의를 포함해야 한다(MUST).

20.2 TOSCA 메타 파일

TOSCA 메타 파일은 CSAR에 있는 여러 아티팩트를 적절하게 해석할 수 있도록 하는 메타 파일을 포함한다. CSAR의 TOSCA-Metadata 디렉터리는 TOSCA.meta 파일을 포함한다.

TOSCA 메타 파일은 이름/값의 쌍으로 구성되는데 이는 이름 부분 다음에 콜론, 공백 그리고 값 부분을 삽입하여 표기한다. 이름은 콜론을 포함해서는 안 되고(MUST NOT) 이진 데이터 값은 'base64' 인코딩을 해야 한다(MUST). 한 줄을 넘는 값의 경우 두 번째 줄부터는 최소한 하나 이상의 공백으로 시작해야 한다(값 문자열을 읽어 들일 때, 해당 공백은 없어지므로 문제없음).

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

각 이름/값의 쌍은 다른 줄에 표시한다. 관련된 이름/값 쌍의 목록(즉 CSAR에서 특정 파일을 서술하는 연속적인 이름/값 쌍의 목록)을 *블록(block)*이라 부른다. 빈 줄로 블록을 분리하고 '*block_0*'라 부르는 첫 번째 블록은 CSAR 자체에 대한 메타데이터이고 나머지 다른 블록은 CSAR 안에 있는 파일들의 메타데이터를 나타낸다.
TOSCA 메타파일에 있는 'block_0'의 구조는 다음과 같다.

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

상기 이름/값의 쌍에 대한 설명은 다음과 같다.

- TOSCA-Meta-File-Version: 이는 TOSCA 메타파일 포맷의 버전 번호이다. TOSCA 명세의 현재 버전에서 그 값은 "1.0"이어야 한다(MUST).

- CSAR-Version: CSAR 명세의 버전 번호이다. TOSCA 명세의 현재 버전에서 그 값은 "1.0"이어야 한다(MUST).

- Created-By: CSAR을 만드는 사람이나 공급자

- Entry-Definitions: 이 선택적(OPTIONAL) 이름/값의 쌍은 CSAR의 Definitions 디렉터리에 있는 TOSCA 정의서 파일을 가리키며 이는 CSAR의 콘텐츠를 처리하기 위한 진입점으로써 사용하여야 한다(SHOULD). 단, CSAR이 다수의 정의서 파일을 포함할 수 있다는 것에 주의한다. 이는 완전성을 위한 것으로, 정의서 파일 중 하나에서 정의한 서비스 템플릿은 외부에서 들여올 필요 없이 Definitions 디렉터리 내 다른 정의서 파일에서 정의한 노드 유형을 참조하는 것을 가능하게 한다. Entry-Definitions 이름/값의 쌍은 Definitions 디렉터리에 있는 일련의 파일을 최적으로 처리하기 위한 힌트를 제공한다.

('block_0'이 외의) 블록에서 첫 번째 행은 "Name"을 이름으로 갖고 서술하는 파일의 경로명을 값으로 갖는 이름/값 쌍이어야 한다(MUST). 두 번째 행은 "Content-Type"을 이름으로 갖고 서술한 파일의 유형을 값('type/sub_type' 형식의 MIME 유형)으로 갖는 이름/값 쌍이어야 한다(MUST). 나머지의 연속된 행의 이름/값 쌍은 파일-유형에 따라 다르다.

```
Name: <path-name_1>
Content-Type: type_1/subtype_1
<name 11>: <value 11>
<name_12>: <value_12>
...
<name 1n>: <value 1n>

...

 Name: <path-name k>
 Content-Type: type_k/subtype_k
 <name_k1>: <value_k1>
 <name k2>: <value k2>
 ...
 <name_km>: <value_km>
```

상기 이름/값의 쌍에 대한 설명은 다음과 같다.

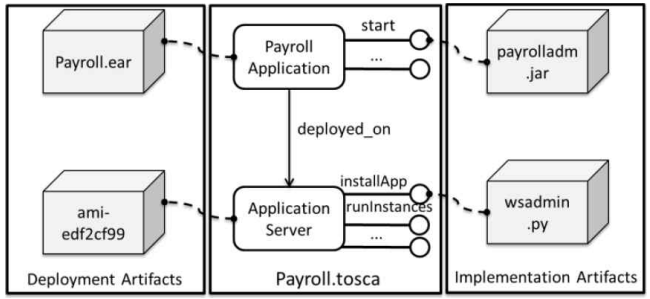
- Name: 실제 CSAR 내에서 서술하는 파일이나 자원의 경로명 또는 경로명 패턴. 단, 해당 위치에 있는 파일이 기본적으로 외부 파일에 대한 참조를 포함할 수도 있다(MAY). URL 스키마 "file", "http" 또는 "https" 중 하나인 URI가 그런 참조를 제공한다.

- Content-Type: 서술한 파일의 유형. 이 유형은 'type/sub_type' 형식을 준수하는 MIME 유형이다. 공급자가 정의한 서브 유형은 "vnd." 문자열로 시작하여야 한다(SHOULD).

후속 지시어가 이전의 지시어에 재정의 한다는 것에 주의한다. 이는 전반적인 기본 지시어를 명시하고 TOSCA 메타파일의 후속 지시어를 통하여 값을 특화할 수 있게 한다.

20.3 예시

(그림 20-1)은 어느 애플리케이션의 서비스 템플릿을 포함하는 Payroll.tosca라는 이름의 예시 정의서 파일이다. 해당 애플리케이션은 Java로 작성된 급여 장부 애플리케이션으로 적절한 애플리케이션 서버에 배포하여야 한다(MUST). 해당 애플리케이션의 서비스 템플릿은 노드 템플릿인 Payroll Application, 노드 템플릿인 Application Server 및 관계 템플릿인 deployed_on을 정의한다. 해당 Payroll Application은 Payroll Application 노드 템플릿의 배포 아티팩트로 제공되는 Payroll.ear라는 이름의 EAR 파일과 결합한다. 아마존 머신 이미지(Amazon Machine Image, AMI)는 Application Server 노드 템플릿의 배포 아티팩트이다. 해당 배포 아티팩트는 Amazon EC2 환경에 있는 이미지에 대한 참조이다. 노드 템플릿의 어떤 동작에 대한 구현 아티팩트도 제공된다. 예를 들어 Payroll Application의 start 동작은 payrolladm.jar 파일이 지원하는 Java API를 사용하여 구현하고 runInstances 동작이 AMI의 인스턴스를 실행하기 위하여 Amazon에서 사용할 수 있는 REST API인 반면에 Application Server의 installApp 동작은 Python 스크립트인 wsadmin.py를 사용하여 구현한다. runInstances 동작은 Amazon Web Service(https://ec2.amazonaws.com/?Action=RunInstances)를 통해 사용할 수 있기 때문에 특정 구현 아티팩트와 관련이 없다는 것에 주의한다. 다만, Application Server 노드 유형의 동작으로 이 REST API에 대한 상세한 사항을 명시한다.



(그림 20-1) 예시 서비스 템플릿

PayrollTypes.tosca 문서에서 해당 노드 유형과 관계 유형을 정의하고 있으며 Payroll 서비스 템플릿을 포함하는 정의서가 PayrollTypes.tosca를 불러온다. 다음은 해당 정의서의 일부 내용을 나타낸다.

```
<Definitions id="PayrollDefinitions"
             targetNamespace="http://www.example.com/tosca"
             xmlns:pay="http://www.example.com/tosca/Types">

  <Import namespace="http://www.example.com/tosca/Types"
          location="http://www.example.com/tosca/Types/PayrollTypes.tosca"
          importType=" http://docs.oasis-open.org/tosca/ns/2011/12"/>

  <Types>
     ...
```

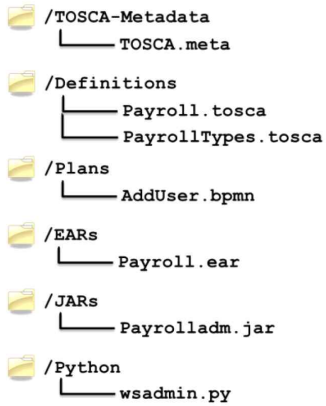


```
  </Types>

  <ServiceTemplate id="Payroll" name="Payroll Service Template">

    <TopologyTemplate ID="PayrollTemplate">

      <NodeTemplate id="Payroll Application"
                    type="pay:ApplicationNodeType">
        ...

        <DeploymentArtifacts>
          <DeploymentArtifact name="PayrollEAR"
                              type="http://www.example.com/
                                    ns/tosca/2011/12/
                                    DeploymentArtifactTypes/CSARref">
                EARs/Payroll.ear
          </DeploymentArtifact>
        </DeploymentArtifacts>

      </NodeTemplate>

      <NodeTemplate id="Application Server"
                    type="pay:ApplicationServerNodeType">
        ...

        <DeploymentArtifacts>
          <DeploymentArtifact name="ApplicationServerImage"
                              type="http://www.example.com/
                                    ns/tosca/2011/12/
                                    DeploymentArtifactTypes/AMIref">
                ami-edf2cf99
          </DeploymentArtifact>
        </DeploymentArtifacts>

      </NodeTemplate>

      <RelationshipTemplate id="deployed on"
                            type="pay:deployed_on">
         <SourceElement ref="Payroll Application"/>
         <TargetElement ref="Application Server"/>
      </RelationshipTemplate>

    </TopologyTemplate>

   </ServiceTemplate>

</Definitions>
```

Payroll Application 노드 템플릿은 배포 아티팩트인 PayrollEAR를 명시한다. 이는 Payroll.tosca 파일을 포함하는 CSAR에 대한 참조이며 이는 DeploymentArtifact 요소의 "**.../CSARref**"라는 유형 값을 통하여 나타난다. 유형 특화 콘텐츠는 CSAR의 디렉터리 구조에 있는 경로 표현식이다. 이는 CSAR의 EARs 디렉터리에 있는 Payroll.ear 파일을 가리킨다(해당 CSAR의 구조에 대해서는 (그림 19-2) 참고).

Application Server 노드 템플릿은 아마존 머신 이미지(AMI)를 참조하는 ApplicationServerImage라는 이름의 배포 아티팩트를 가지고 있으며 ".../AMIref"라는 유형 값을 통하여 이를 표시한다.

해당 CSAR은 (그림 19-2)과 같은 구조를 가진다. TOSCA-Metadata 디렉터리에 TOSCA.meta 파일이 있다. Payroll.tosca 파일 자체는 Definitions 디렉터리에 있다. 그리고 PayrollTypes.tosca 파일은 이 디렉터리에 있다. 다른 디렉터리의 내용에 대해서는 위에서 이미 간략하게 설명하였다.



(그림 19-2) 예시 CSAR의 구조

TOSCA.meta 파일은 다음과 같다.

```
1  TOSCA-Meta-Version: 1.0
2  CSAR-Version: 1.0
3  Created-By: Frank
4
5  Name: Service-Template/Payroll.tosca
6  Content-Type: application/vnd.oasis.tosca.definitions
7
8  Name: Service-Template/PayrollTypes.tosca
9  Content-Type: application/vnd.oasis.tosca.definitions
10
11 Name: Plans/AddUser.bpmn
12 Content-Type: application/vnd.oasis.bpmn
13
14 Name: EARs/Payroll.ear
15 Content-Type: application/vnd.oasis.ear
16
17 Name: JARs/Payrolladm.jar
18 Content-Type: application/vnd.oasis.jar
19
20 Name: Python/wsadmin.py
21 Content-Type: application/
```


21 보안 고려사항

TOSCA는 사용자 인증을 위해 특정 방식이나 기술의 사용을 강제하지는 않는다. 하지만 사용자가 한 가지 보안 원칙을 제시하거나 인프라스트럭처에서 해당 원칙을 반드시 지원해야한다.

부 록 Ⅰ-1

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

지식재산권 확약서 정보

Ⅰ-1.1 지식재산권 확약서

해당 사항 없음

※ 상기 기재된 지식재산권 확약서 이외에도 본 표준이 발간된 후 접수된 확약서가 있을 수 있으니, TTA 웹사이트에서 확인하시기 바랍니다.

부 록 Ⅰ-2

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

시험인증 관련 사항

Ⅰ-2.1 시험인증 대상 여부

해당 사항 없음

Ⅰ-2.2 시험표준 제정 현황

해당 사항 없음

부 록 Ⅰ-3

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

본 표준의 연계(family) 표준

Ⅰ-3.1 TTAK.xx-xx.xxxx/R1 클라우드 애플리케이션을 위한 YAML 형식의 TOSCA 간략 프로파일

TTAK.xx-xx.xxxx/R1 표준은 TOSCA 서비스 템플릿의 저작을 단순화하기 위한 YAML[1] 구문으로 TOSCA v1.0 설명서의 간략 프로파일을 정의한다.

1) YAML은 XML보다 읽거나 편집하기가 훨씬 쉬운 인간 친화적인 데이터 직렬화 표준이다(http://yaml.org/).

부 록 Ⅰ-4

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

참고 문헌

해당 사항 없음

부 록 Ⅰ-5

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

영문표준 해설서

해당 사항 없음

부 록 Ⅰ-6

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

표준의 이력

판수	채택일	표준번호	내용	담당 위원회
제1판	2019.0x.xx	제정 TTAK.xx-xx.xxxx	-	클라우드 컴퓨팅 프로젝트그룹 (PG1003)