(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2011년04월04일
(11) 등록번호 10-1026606
(24) 등록일자 2011년03월25일

(51) Int. Cl.
G06F 9/00 (2006.01)
(21) 출원번호 10-2004-0015016
(22) 출원일자 2004년03월05일
심사청구일자 2009년02월02일
(65) 공개번호 10-2004-0079317
(43) 공개일자 2004년09월14일
(30) 우선권주장
60/452,736 2003년03월06일 미국(US)
10/693,838 2003년10월24일 미국(US)
(56) 선행기술조사문헌
US06336138 B1*
US20020198995 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
헌트갈렌씨.
미국98008워싱턴주벨뷰162번애비뉴에스이2967
오드레드제프리
미국98103워싱턴주시애틀배그레이애비뉴엔.4028
(뒷면에 계속)
(74) 대리인
주성민, 이중희, 백만기

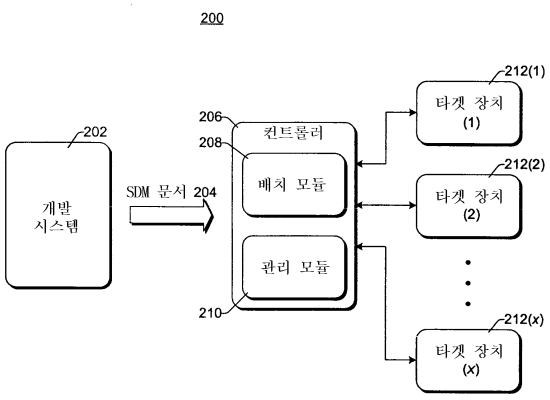
전체 청구항 수 : 총 34 항 심사관 : 복진요

(54) 시스템용 통합 설계, 배치 및 관리방법, 장치, 시스템 및 컴퓨터 판독가능 기록매체

(57) 요 약

소정의 형태에 따라 시스템에 대한 통합 설계, 배치 및 관리 페이즈는 시스템 정의 모델을 사용하여 시스템을 설계하는 것을 포함한다. 시스템 정의 모델은 그후 하나 이상의 컴퓨팅 장치에 상기 시스템을 배치하는 데 사용되고, 시스템이 배치된 후, 시스템 정의 모델은 하나 이상의 컴퓨팅 장치 상에 배치된 시스템을 관리하는 데 사용된다.

대 표 도 - 도2



(72) 발명자

다바라바쌈

미국98107워싱턴주시애틀배이커애비뉴엔더블유4119

그릴리시케빈

미국98103워싱턴주시애틀데이튼애비뉴엔.4035

멘칭롭

미국98052워싱턴주레드몬드164번애비뉴엔이10507

특허청구의 범위

청구항 1

하나 이상의 컴퓨팅 장치에 의해 시스템의 개발 페이즈(phase)에서 시스템 정의 모델(system definition model)을 사용하여 상기 시스템을 설계하는 단계 - 상기 시스템은 애플리케이션이고, 상기 시스템을 설계하는 단계는, 상기 시스템 정의 모델 내에서 상기 시스템이 환경에서 실행되기 위해 상기 환경에 의해 만족되어야 하는 제한을 포함하는 단계를 포함함 -;

후속적으로 상기 하나 이상의 컴퓨팅 장치에 의해 상기 시스템의 배치 페이즈에서 상기 시스템 정의 모델을 사용하여 상기 시스템을 상기 하나 이상의 컴퓨팅 장치의 적어도 하나 상에 배치(deploy)하는 단계;

상기 시스템의 배치 후에, 상기 하나 이상의 컴퓨팅 장치에 의해, 상기 시스템의 관리 페이즈 중에 상기 시스템 정의 모델 내에서 정의된 하나 이상의 기능을 호출하여 상기 하나 이상의 컴퓨팅 장치의 적어도 하나 상에 배치된 상기 시스템을 관리하는 단계; 및

상기 하나 이상의 컴퓨팅 장치에 의해, 적어도 상기 시스템의 설계 중에 상기 제한이 만족되는 것을 확인하는 단계

를 포함하는 방법.

청구항 2

삭제

청구항 3

삭제

청구항 4

제1항에 있어서,

상기 시스템을 관리하는 동안 얻어진 지식을 이용하여 상기 시스템의 후속 버전을 설계하는 단계를 더 포함하는 방법.

청구항 5

제1항에 있어서,

상기 시스템 정의 모델은 상기 시스템을 상기 하나 이상의 컴퓨팅 장치 상에 배치하는 방법을 설명하는 지식을 포함하는 방법.

청구항 6

제1항에 있어서,

상기 시스템 정의 모델은 상기 시스템을 다수의 상이한 컴퓨팅 장치 상에 배치하는 방법을 설명하는 지식을 포함하고, 상기 지식은 상기 시스템을 상기 다수의 상이한 컴퓨터 장치의 각각에 배치하는 방법을 설명하는 다른 지식을 포함하는 방법.

청구항 7

삭제

청구항 8

제1항에 있어서,

상기 시스템 정의 모델은 상기 시스템의 설계 동안 상기 하나 이상의 컴퓨팅 장치에 의해 상기 제한이 만족되는지를 검사하기 위하여 사용될 수 있는 방법.

청구항 9

제1항에 있어서,

상기 시스템 정의 모델은 상기 시스템의 설계 동안 및 상기 시스템의 관리 동안 상기 하나 이상의 컴퓨팅 장치에 의해 상기 제한이 만족되는지를 검사하기 위하여 사용될 수 있는 방법.

청구항 10

제1항에 있어서,

상기 시스템 정의 모델은 상기 시스템의 배치 후에 상기 시스템을 관리하는 방법을 설명하는 지식을 포함하는 방법.

청구항 11

제1항에 있어서,

상기 시스템의 관리 동안, 플로우를 사용하여 상기 시스템에 구성 변경(configuration change)을 자동적으로 전파하는 단계를 더 포함하는 방법.

청구항 12

제1항에 있어서,

상기 방법은, 상기 시스템의 설계, 배치, 및 관리 이전에,

상기 하나 이상의 컴퓨팅 장치에 의해, 다른 시스템 정의 모델을 사용하여 상기 환경을 설계하는 단계 - 상기 시스템은 상기 하나 이상의 컴퓨팅 장치 상의 상기 환경에 배치됨 -;

상기 하나 이상의 컴퓨팅 장치에 의해, 후속적으로 상기 다른 시스템 정의 모델을 사용하여, 상기 환경을 상기 하나 이상의 컴퓨팅 장치 상에 배치하는 단계; 및

상기 환경의 배치 후에, 상기 하나 이상의 컴퓨팅 장치에 의해, 상기 다른 시스템 정의 모델을 사용하여 상기 하나 이상의 컴퓨팅 장치 상에 배치된 환경을 관리하는 단계를 더 포함하고,

상기 시스템 정의 모델은, 상기 시스템이 상기 하나 이상의 컴퓨팅 장치 상에서 실행되도록 하기 위하여, 상기 환경에 의해 만족되어야 하는 제한을 포함하고, 상기 다른 시스템 정의 모델은, 상기 시스템이 상기 하나 이상의 컴퓨팅 장치 상에서 실행되도록 하기 위하여, 상기 시스템에 의해 만족되어야 하는 다른 제한을 포함하는 방법.

청구항 13

제12항에 있어서,

상기 환경에 대한 상기 시스템 정의 모델은 상기 하나 이상의 컴퓨팅 장치의 구성의 조사를 통해 도출되는 방법.

청구항 14

삭제

청구항 15

제1항에 있어서,

상기 하나 이상의 컴퓨팅 장치 상에 복수의 환경이 배치되고,

상기 방법은,

복수의 상이한 시스템 정의 모델을 사용하여 상기 복수의 환경의 각각을 설계하는 단계 - 상기 복수의 환경의 각각은 상기 복수의 상이한 시스템 정의 모델 중의 하나와 관련됨 -;

각각의 환경에 대하여, 상기 복수의 상이한 시스템 정의 모델 중의 관련된 시스템 정의 모델을 사용하여 상기

환경을 배치하는 단계; 및

배치 후에, 각각의 환경에 대하여, 상기 복수의 상이한 시스템 정의 모델 중의 관련된 시스템 정의 모델을 사용하여 상기 환경을 관리하는 단계

를 더 포함하는 방법.

청구항 16

제15항에 있어서,

상기 복수의 환경의 각각은 계층화되고, 상기 복수의 환경의 각각은 상기 복수의 환경 중 다른 환경 또는 상기 시스템에 대한 환경으로서 기능하는 방법.

청구항 17

복수 개의 명령어들이 기록된 하나 이상의 컴퓨터 판독가능 기록매체로서,

상기 명령어들은 프로세서에 의해 실행될 때, 상기 프로세서로 하여금,

시스템의 개발 페이즈에서 시스템 정의 모델을 사용하여 상기 시스템을 설계하고 - 상기 시스템은 애플리케이션이고, 상기 시스템 정의 모델은 상기 애플리케이션이 배치될 환경의 표시를 포함하고, 상기 시스템을 설계하는 것은, 상기 애플리케이션을 상기 시스템 정의 모델 내의 상기 표시로 바인딩하는 것을 포함하고, 상기 표시는 그 애플리케이션 컴포넌트의 환경 호스트에 대한 정의 및 그 애플리케이션의 구성 상의 제한을 포함함 -,

후속적으로 시스템의 배치 페이즈에서 상기 시스템 정의 모델을 사용하여, 상기 시스템을 하나 이상의 컴퓨팅 장치 상에 배치하고,

상기 시스템의 배치 후에, 상기 시스템의 관리 페이즈에서 상기 시스템 정의 모델 내에서 정의된 하나 이상의 기능을 호출하여 상기 하나 이상의 컴퓨팅 장치 상에 배치된 상기 시스템을 관리하도록 하는

하나 이상의 컴퓨터 판독가능 기록매체.

청구항 18

삭제

청구항 19

삭제

청구항 20

제17항에 있어서,

상기 시스템 정의 모델은 상기 시스템을 배치하는 방법을 설명하는 지식을 포함하는 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 21

제17항에 있어서,

상기 시스템 정의 모델은 상기 시스템을 복수의 상이한 환경에 배치하는 방법을 설명하는 지식을 포함하고, 상기 지식은 상기 시스템을 상기 복수의 상이한 환경의 각각에 배치하는 방법을 설명하는 상이한 지식을 포함하는 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 22

제17항에 있어서,

상기 시스템 정의 모델은 상기 시스템이 환경에서 실행되도록 하기 위하여 상기 환경에 의해 만족되어야 하는 제한을 포함하는 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 23

제22항에 있어서,

상기 시스템 정의 모델을 사용하여 상기 시스템을 배치하는 것은 상기 시스템 정의 모델을 사용하여 상기 시스템의 설계 동안 상기 환경에 의해 상기 제한이 만족되는지를 검사하는 것인 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 24

제17항에 있어서,

상기 시스템 정의 모델은 상기 시스템을 관리하는 방법을 설명하는 지식을 포함하는 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 25

프로세서;

시스템의 개발 페이즈에서 시스템 정의 모델을 사용하여 상기 시스템을 설계하기 위한, 상기 프로세서에 의해 동작되는 수단 - 상기 시스템은 애플리케이션이고, 상기 시스템의 개발 페이즈에서 시스템 정의 모델을 사용하여 상기 시스템을 설계하는 것은, 상기 시스템 정의 모델 내에서 상기 시스템이 환경에서 실행되기 위해 상기 환경에 의해 만족되어야 하는 제한을 포함하는 것을 포함함 -;

후속적으로 상기 시스템의 배치 페이즈에서 상기 시스템 정의 모델을 사용하여 상기 시스템을 하나 이상의 컴퓨팅 장치 상에 배치하기 위한, 상기 프로세서에 의해 동작되는 수단;

상기 시스템의 배치 후에, 상기 시스템의 관리 페이즈에서 상기 시스템 정의 모델 내에서 정의된 하나 이상의 기능을 호출하여 상기 하나 이상의 컴퓨팅 장치 상에 배치된 상기 시스템을 관리하기 위한, 상기 프로세서에 의해 동작되는 수단; 및

적어도 상기 시스템의 설계 중에 상기 제한이 만족되는 것을 확인하기 위한, 상기 프로세서에 의해 동작되는 수단

을 포함하는 장치.

청구항 26

제25항에 있어서,

상기 후속적으로 상기 시스템의 배치 페이즈에서 상기 시스템 정의 모델을 사용하여 배치하기 위한 수단은 상기 시스템을 배치하는 방법을 설명하는 지식을 상기 시스템 정의 모델 내에 포함시키는 수단을 포함하는 장치.

청구항 27

제25항에 있어서,

상기 후속적으로 상기 시스템의 배치 페이즈에서 상기 시스템 정의 모델을 사용하여 배치하기 위한 수단은 상기 시스템을 복수의 상이한 환경에 배치하는 방법을 설명하는 지식을 상기 시스템 정의 모델 내에 포함시키는 수단을 포함하고,

상기 지식은 상기 시스템을 상기 복수의 상이한 환경의 각각에 배치하는 방법을 설명하는 상이한 지식을 포함하는 장치.

청구항 28

삭제

청구항 29

삭제

청구항 30

제25항에 있어서,

상기 관리 페이즈에서 상기 시스템 정의 모델을 사용하여 관리하기 위한 수단은 상기 시스템을 관리하는 방법을 설명하는 지식을 상기 시스템 정의 모델 내에 포함시키는 수단을 포함하는 장치.

청구항 31

프로세서; 및

복수 개의 실행가능한 명령어들을 포함하고,

상기 명령어들은 상기 프로세서에 의해 실행될 때 작업들을 수행하고,

상기 작업들은

시스템 정의 모델을 사용하여 애플리케이션을 설계하는 단계 - 상기 시스템 정의 모델은 상기 애플리케이션의 라이프사이클에 걸쳐 적용가능하고, 상기 애플리케이션의 상기 라이프사이클은 상기 애플리케이션의 설계, 상기 애플리케이션의 배치, 및 상기 애플리케이션의 관리를 포함하고, 상기 시스템 정의 모델은 상기 애플리케이션이 배치될 환경의 표시를 포함하고, 상기 시스템을 설계하는 단계는, 상기 애플리케이션을 상기 시스템 정의 모델 내의 상기 표시로 바인딩하고, 상기 표시는 그 애플리케이션 컴포넌트의 환경 호스트에 대한 정의 및 그 애플리케이션의 구성 상의 제한을 포함함 -;

후속적으로 상기 시스템 정의 모델을 사용하여 상기 시스템을 하나 이상의 컴퓨팅 장치 상에 배치하는 단계;

상기 시스템의 배치 후에 상기 시스템 정의 모델 내에서 정의된 하나 이상의 기능을 호출하여 상기 하나 이상의 컴퓨팅 장치 상에 배치된 상기 애플리케이션을 관리하는 단계

를 포함하고,

상기 시스템은 상기 시스템 정의 모델 내의 기능 동작이 지정되는 방법을 지시하는 스키마를 더 포함하는 시스템.

청구항 32

제31항에 있어서,

상기 시스템 정의 모델은 상기 애플리케이션을 배치하는 방법을 설명하는 정보를 포함하는 시스템.

청구항 33

제31항에 있어서,

상기 시스템 정의 모델은 상기 애플리케이션을 복수의 상이한 환경에 배치하는 방법을 설명하는 정보를 포함하고, 상기 정보는 상기 애플리케이션을 상기 복수의 상이한 환경의 각각에 배치하는 방법을 설명하는 다른 정보를 포함하는 시스템.

청구항 34

제31항에 있어서,

상기 시스템 정의 모델은 상기 애플리케이션이 환경에서 실행되도록 하기 위하여 상기 환경에 의해 만족되어야 하는 제한을 포함하는 시스템.

청구항 35

제34항에 있어서,

상기 시스템 정의 모델은, 상기 애플리케이션의 설계 동안 및 상기 애플리케이션의 관리 동안, 상기 제한이 상기 시스템에서 하나 이상의 컴퓨팅 장치에 의해 만족되는지를 검사하는 데 사용될 수 있는 시스템.

청구항 36

제34항에 있어서,

상기 시스템 정의 모델은, 상기 애플리케이션의 설계 동안 상기 제한이 상기 환경에 의해 만족되는지를 검사하는 데 사용될 수 있는 시스템.

청구항 37

제31항에 있어서,

상기 시스템 정의 모델은 상기 애플리케이션을 관리하는 방법을 설명하는 정보를 포함하는 시스템.

청구항 38

제31항에 있어서,

상기 시스템은,

환경의 라이프사이클에 걸쳐 적용가능한 다른 시스템 정의 모델 - 상기 환경의 라이프사이클은 상기 환경의 설계, 상기 환경의 배치 및 상기 환경의 관리를 포함함 -

을 더 포함하고,

상기 스키마는 상기 다른 시스템 정의 모델 내의 기능 동작이 지정되는 방법을 지시하는 시스템.

청구항 39

제38항에 있어서,

상기 환경에 대한 상기 다른 시스템 정의 모델은 하나 이상의 컴퓨팅 장치의 구성의 조사를 통해 도출되는 시스템.

청구항 40

제38항에 있어서,

상기 시스템 정의 모델은 상기 애플리케이션이 상기 환경 상에서 실행되도록 하기 위하여 상기 환경에 의해 만족되어야 하는 제한을 포함하고, 상기 다른 시스템 정의 모델은 상기 애플리케이션이 상기 환경 상에서 실행되도록 하기 위하여 상기 애플리케이션에 의해 만족되어야 하는 다른 제한을 포함하는 시스템.

청구항 41

제38항에 있어서,

상기 시스템은,

추가의 환경의 라이프사이클에 걸쳐 적용가능한 추가의 시스템 정의 모델 - 상기 추가의 환경의 라이프사이클은 상기 추가의 환경의 설계, 상기 추가의 환경의 배치 및 상기 추가의 환경의 관리를 포함함 -

을 더 포함하고,

상기 추가의 환경은 상기 환경의 하부에 계층화되고,

상기 스키마는 상기 추가의 시스템 정의 모델 내의 기능 동작이 지정되는 방법을 지시하는 시스템.

청구항 42

삭제

청구항 43

삭제

청구항 44

제1항에 있어서,

상기 시스템 정의 모델은 복수의 상이한 런타임에 상기 시스템을 배치하는 방법을 설명하는 정보를 포함하고, 상기 정보는 상기 복수의 상이한 런타임 각각에 상기 시스템을 배치하는 방법을 설명하는 상이한 정보를 포함하는 방법.

청구항 45

삭제

청구항 46

삭제

청구항 47

삭제

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

[0030] 본 발명은 분산형 컴퓨팅 시스템에 대한 아키텍쳐에 관한 것이다.

[0031] 지난 수년간 인터넷 사용은 폭발적으로 증가하였으며 계속적으로 성장하고 있다. 사람들은 전자 메일, 온라인 쇼핑, 뉴스 및 정보 수집, 음악 감상, 비디오 클립 관람, 구직 등의 월드 와이드 웹(또는 간략히, "웹") 상에 제공되는 많은 서비스에 만족하게 되었다. 인터넷 기반 서비스에 대한 성장 요구에 맞추어, 호스팅 웹사이트에 사용되며 이들 사이트에 대한 백엔드 서비스(backend service)를 제공하고 그 사이트와 관련된 데이터를 저장하는 컴퓨터 시스템이 급격히 성장하여 왔다.

[0032] 분산형 컴퓨터 시스템의 하나의 형태는 네트워크 기반 서비스를 호스팅하는 많은 컴퓨터를 하우징하는 특수 설계된 컴플렉스인 {인터넷 데이터 센터(IDC) 등의} 데이터 센터 또는 엔터프라이즈 데이터 센터(EDC)이다. "웹 팜(Webfarm)" 또는 "서버 팜(sever farm)"이라고도 지칭될 수 있는 데이터 센터는 통상 분위기 제어된(climate-controlled) 물리적으로 안전한 빌딩에서 수백 내지 수천의 컴퓨터를 하우징한다. 데이터 센터는 통상 신뢰성 있는 인터넷 액세스, 신뢰성있는 전력 공급 및 안전한 오퍼레이팅 환경을 제공한다.

[0033] 현재, 큰 데이터 센터는 복잡하고, 종종 다수의 애플리케이션을 호스팅하기 위하여 호출된다. 예를 들어, 어떤 웹 사이트는 수천 컴퓨터를 동작시킬 수 있고 많은 분산형 애플리케이션을 호스팅할 수 있다. 이들 분산형 애플리케이션은, 종종 오퍼레이터가 컴퓨터를 소정의 네트워크 스위치에 물리적으로 접속할 뿐만 아니라 데이터 센터 내에 유선 구성을 수동으로 배열하여 복잡한 애플리케이션을 지원하도록 하는 것을 요구하는 복잡한 네트워킹 요구사항을 갖는다. 결과적으로, 애플리케이션 요구사항에 따르도록 물리적 네트워크 토폴로지(topology)를 확립하는 이 태스크는 사람의 실수가 발생하기 쉽고 성가시며 시간을 소비하는 프로세스일 수 있다. 따라서, 물리적 컴퓨팅 시스템 상에 분산형 애플리케이션을 설계하고 배치하는 향상된 기술이 필요하다.

발명이 이루고자 하는 기술적 과제

[0034] 시스템용 통합 설계, 배치 및 관리 페이즈가 설명된다.

[0035] 소정의 양태에 따르면, 시스템 정의 모델은 시스템을 설계하는 데 사용된다. 그후, 시스템 정의 모델은 하나 이상의 컴퓨팅 시스템 상에 시스템을 배치하는 데 사용된다. 시스템의 배치후, 시스템 정의 모델은 하나 이상의 컴퓨팅 장치 상에 배치된 시스템을 관리하는 데 사용된다.

발명의 구성 및 작용

[0036] 다음의 개시는 대규모 애플리케이션 서비스를 갖는 분산형 컴퓨팅 시스템을 설계하고 구현하는 아키텍쳐에 적합한 다수의 양태를 기재한다. 본 명세서는 서비스 정의 모델(SDM)로도 지칭될 수 있는 시스템 정의 모델(SDM) 및 SDM 런타임 환경에 대한 설명을 포함한다. SDM은 추상적인 방법으로 분산형 컴퓨터 애플리케이션과 데이터

센터를 설계하기 위한 애플리케이션 설계자를 위한 컨텍스트(context)와 툴을 제공한다. 모델은 물리적 컴퓨터 리소스와 소프트웨어에 의해 결국 구현될 애플리케이션의 기능 유닛을 나타내는 요소의 세트를 정의한다. 모델 요소는 컴포넌트에 의해 표시되는 기능 동작이 어떻게 지정되는지를 지시하는 스키마(schema)와 관련된다.

[0037] 여기에서 이용된 바와 같이, 용어 "유선(wire)"은 또한 "접속", "통신" 또는 "통신 관계"로도 지칭될 수 있다. 또한, 용어 "시스템"은 "모듈"로도 지칭될 수 있으며, 용어 "리소스 공간"은 "리소스"로 지칭될 수 있다. 또한, 용어 "애플리케이션 공간"은 "애플리케이션"으로도 지칭되고, 용어 "인스턴스 공간"은 "인스턴스"로도 지칭될 수 있다. 또한, 용어 "클래스"는 "추상 정의"로도 지칭되고, 용어 "포트"는 또한 "엔드포인트"로 지칭되고, 용어 "타입"은 "정의"로 지칭될 수 있다.

[0038] 도 1은 네트워크 세팅(100)의 일례를 나타낸다. 세팅(100)에서, 다수(x)의 컴퓨팅 장치{102(1), 102(2), ..., 102(x)}는 네트워크(106)에 결합된다. 네트워크(106)는 (유선 및/또는 무선 네트워크를 포함하여) 종래의 다양한 네트워크 토폴로지 및 타입 중의 임의의 것을 표시하고, (공중 및/또는 개인 프로토콜을 포함하여) 종래의 다양한 네트워크 프로토콜 중의 임의의 것을 사용하는 것으로 한다. 네트워크(106)는 예를 들어, 근거리 통신망(LAN), 원거리 통신망(WAN), 인터넷의 부분 등을 포함할 수 있다. 세팅(100)은 예를 들어 데이터 센터{예를 들어, 인터넷 데이터 센터(IDC)}, 오피스 또는 비즈니스 세팅, 홈 세팅, 교육 또는 연구 설비, 소매 또는 판매 세팅, 데이터 저장 세팅 등을 포함하는 다양한 세팅 중의 임의의 것을 나타낸다.

[0039] 컴퓨팅 장치(102)는 데스크탑 PC, 워크스테이션, 메인프레임 컴퓨터, 서버 컴퓨터, 인터넷 장비(Internet appliance), 게임 콘솔, 핸드헬드 컴퓨터, 셀룰러 전화, 개인 휴대용 단말기(PDA)를 포함하는 종래의 다양한 컴퓨팅 장치 중의 임의의 것일 수 있다. 하나 이상의 장치(102)는 동일 타입의 장치 또는 다른 타입의 장치일 수 있다. 또한, 다수의 장치가 동일 타입의 장치이더라도, 다수의 장치가 다르게 구성될 수 있다 {예를 들어, 2개의 장치(102)가 서버 컴퓨터일 수 있지만, 다른 프로세서, 다른 용량의 RAM, 다른 사이즈의 하드 디스크 드라이브 등의 다른 하드웨어 구성을 가질 수 있다}.

[0040] 하나 이상의 컴퓨팅 장치(102)는 또한 세팅(100)에 부가된 후 재구성될 수 있다. 예를 들어, 특정 컴퓨팅 장치(102)는 하나의 기능을 수행하는 기간(예를 들어, 분, 시, 일, 달 등)동안 동작할 수 있고, 그후, 관리자가 다른 기능이 바람직하다는 것을 결정할 수 있다(예를 들어, 서버 컴퓨터로부터 워크스테이션 컴퓨터로의 변경, 웹 서버로부터 로컬 파일 서버로의 변경 등).

[0041] 도 2는 시스템 정의 모델을 사용하는 아키텍처(200)의 일례를 나타내는 블록도이다. SDM은 시스템의 전체 라이프사이클에 걸쳐 사용되도록 설계된다. 시스템은, 함께 작업하여 공통 기능을 달성할 수 있는 관련된 소프트웨어 및/또는 하드웨어 리소스의 세트이다. 이러한 시스템의 일례는, 컴퓨팅 시스템에 의해 운영되거나 실행되어 다양한 기능을 수행할 수 있는 명령의 세트로 지칭되는 애플리케이션이다. 애플리케이션의 예는 게임과 같은 엔터테인먼트 애플리케이션, 워드 프로세서와 같은 생산성 애플리케이션, 전자 사전과 같은 참조 애플리케이션, 웹 서비스 또는 금융 분석에 사용될 수 있는 분산형 애플리케이션 등을 포함한다. 이러한 시스템의 다른 예는 애플리케이션(또는 또다른 환경)이 배치될 수 있는 환경이다. 환경은 애플리케이션(또는 또다른 환경)이 배치되는 소프트웨어 및/또는 하드웨어 리소스를 지칭한다. 이러한 환경은 이하에서 상세히 설명하는 바와 같이 계층화(layered)된다.

[0042] 시스템의 라이프사이클은 통상 3개의 주요 페이즈(스테이지로도 지칭됨), 즉, 설계 또는 개발 페이즈, 그 후의 배치 또는 설치 페이즈, 그 후의 동작 또는 관리 페이즈를 포함한다. 모델이 시스템의 라이프사이클의 세가지 모든 페이즈에 적용됨에 따라, 모델은 시스템의 라이프사이클의 다양한 페이즈에 대한 통합 포인트로서 간주될 수 있고, 이들 페이즈의 각각을 용이하게 한다. 또한, 모델을 사용하여, 이들 페이즈들간에, 시스템의 관리에 대한 지식(예를 들어, 미래의 버전을 위하여 설계 및 개발 팀에 피드백되어 설계 및 개발 팀이 시스템을 변경하도록 하거나, 현재 버전의 성능을 개선하도록 함), 시스템의 구조, 배치 요구사항 및 동작 반응에 관한 지식, 데스크탑으로부터 데이터 센터로의 동작 환경에 대한 지식, 최종 사용자에 의해 관찰되는 서비스 레벨에 대한 지식 등의 지식이 전달될 수 있다.

[0043] 일반적으로, 설계 페이즈 동안, SDM에 영향을 주는 개발 툴이 통신 소프트웨어 및 하드웨어 컴포넌트로 구성된 시스템을 정의하는 데 사용된다. 시스템 정의는 요구된 리소스, 구성, 동작 특징, 정책 등을 포함하는 분산형 시스템을 배치하고 동작하는데 필요한 모든 정보를 포함한다. 배치 페이즈 동안, 시스템 정의는 시스템을 자동적으로 배치하고 요구된 소프트웨어 및 하드웨어(예를 들어, 서버, 기억장치 및 네트워킹) 리소스를 동적으로 할당하고 구성하는 데 사용된다. 동일한 시스템 정의는 다른 호스트 환경 및 다른 스케일로의 배치에 사용될 수 있다. 관리 페이즈 동안, 오퍼레이팅 시스템 내의 SDM 서비스는 시스템을 관리하는 시스템 레벨 뷰(system-

level view)를 제공한다. 이것은 새로운 관리 툴이 리소스 할당, 구성 관리, 업그레이드 및 시스템의 견지로부터 프로세스 자동화의 유도를 가능하게 한다.

[0044] 아키텍처(200)는 SDM 정의 모델 내의 기능 동작을 정의하는 스키마뿐만 아니라 SDM 정의 모델을 사용한다. 정의 모델은 총괄하여 "정의"로 지칭되는 다양한 종류의 데이터 구조를 포함한다. SDM의 기능은 애플리케이션 프로그램 인터페이스(API) 등의 하나 이상의 플랫폼 서비스를 통해 노출된다.

[0045] 시스템에 대한 설계 페이즈 동안, 개발 시스템(202)은 SDM 문서(204) 등의 시스템 정의를 포함하는 문서를 생성한다. 개발 시스템(202)은 워싱턴주 레드몬드의 마이크로소프트 코포레이션(Microsoft® Corporation)으로부터 이용가능한 비주얼 스튜디오(Visual Studio®) 개발 시스템 등의 다양한 개발 시스템 중의 임의의 것일 수 있다. SDM 문서(204)는 시스템의 배치 및 관리에 관련된 모든 정보(지식으로도 지칭됨)를 정의한다. 시스템을 배치하거나 시스템을 관리할 때 사용되거나 필요한 지식은 SDM 문서(204)에 포함된다. 단일 문서로서 여기에 기재되었지만, 지식은 다수의 문서에서 전개되고 유지될 수 있음이 자명하다.

[0046] 시스템 정의는 리소스, 엔드 포인트, 관계 및 서브시스템 중의 하나 이상에 의하여 시스템을 정의한다. 시스템 정의는 SDM 문서(예를 들어, XML 문서)에서 선언된다. 리소스는 하드웨어 리소스 또는 소프트웨어 리소스일 수 있다. 엔드포인트는 시스템에 걸친 통신을 나타낸다. 관계는 시스템, 리소스 및 엔드포인트간의 관련을 정의한다. 서브시스템은 완전한 시스템으로서 취급될 수 있고 통상 더 큰 시스템의 부분이다.

[0047] 시스템 정의는 동적 시스템의 기본 구조를 캡처한다. 이것은 모든 다른 정보가 추가되는 윤곽(skeleton)으로서 간주될 수 있다. 이 구조는 통상 설계자 및 개발자에 의해 개발 프로세스 동안 지정되고, 통상 빈번히 변경되지 않는다. 구조에 더하여, SDM은 배치 정보, 설치 프로세스, 및 구성, 이벤트 및 수단에 대한 스키마, 자동화 태스크, 건강 모델, 동작 정책 등을 포함할 수 있다. 다른 정보는 분산형 시스템의 라이프사이클에 걸쳐 동작 스태프(staff), 판매자, 및/또는 관리 시스템에 의해 부가될 수 있다.

[0048] SDM 문서(204)는, 시스템이 배치되고 및/또는 실행되는 환경이 만족해야 하는 시스템의 하나 이상의 제한(또한 요구사항으로 지칭됨)을 포함한다. 환경 자체는 SDM 문서를 사용하여 기술된다. 이러한 환경은 단일 컴퓨팅 장치일 수 있거나 대안으로 컴퓨팅 장치(예를 들어, 데이터 센터), 애플리케이션 호스트 등의 집합일 수 있다. 다른 시스템은 다른 환경에 설치될 수 있다. 예를 들어, 데이터 센터는 50개의 컴퓨팅 장치를 포함할 수 있고, 하나의 시스템은 이들 컴퓨팅 장치 중의 5개에 배치되고 나머지 시스템이 이들 컴퓨팅 시스템 중의 35개에 배치될 수 있다. 이들 요구 사항은, 시스템이 배치되는 컴퓨팅 장치(들)에 관한 하드웨어 요구사항(예를 들어, 최소 프로세서 속도, 최소량의 메모리, 최소량의 프리 하드 드라이브 공간, 최소량의 이용가능한 네트워크 대역폭, 이용가능한 특정 보안 메커니즘 등), 시스템이 배치되는 컴퓨팅 장치(들)에 관한 소프트웨어 요구사항(예를 들어, 특정 오퍼레이팅 시스템, 또한 설치되어야 하는 하나 이상의 다른 애플리케이션, 특정 시스템 및/또는 오퍼레이팅 시스템이 구성되는 방법에 관한 스펙, 특정 타입의 보안 및 사용의 암호화 등), 시스템이 배치되는 컴퓨팅 장치(들)에 관한 다른 요구사항(예를 들어, 이용가능한 특정 보안 키, 실시되어야 하는 데이터 센터 정책, 사용되는 인증, 환경 토폴로지 등) 등의 다양한 형태를 취할 수 있다.

[0049] 요구사항은 또한 다른 방향으로 향할 수 있다. 즉, 환경은 설치되는 시스템의 구성에 대한 제한 또는 요구사항(예를 들어, 환경의 표준 또는 정책을 구현하기 위함)을 가질 수 있다. 이들은 시스템이 가져야 하는 특정 세팅 또는 구성, 시스템이 제공하거나 지원해야 하는 특정 기능, 및 시스템이 지원해야 하는 특정 보안 메커니즘 등의 환경의 오퍼레이터에 의해 생성된 "명시적(explicit)" 요구사항일 수 있다. 이들은 또한 환경의 특정 구성 때문에 발생하는 "암시적(implicit)" 요구사항일 수 있다. 예를 들어, 그 환경에서의 호스트 컴퓨팅 장치가 특정 타입의 파일 시스템을 사용하면, 일부의 액션이 그 파일 시스템을 사용하여 수행되는 것이 가능하지 않을 수 있다(비록 그 동일한 액션이 다른 파일 시스템을 사용하여 수행될 수 있을지라도).

[0050] 시스템의 설계 및 개발 페이즈 동안, SDM 문서(204)는 하나 이상의 특정 환경(들)에 대한 시스템을 유효화(validate)하는 데 사용될 수 있다. 이것은 시스템이 환경에 대하여 유효화되는 것과 환경이 시스템에 대하여 유효화되는 2가지의 유효화가 있다. 환경은 SDM 문서(204)에서 식별된 요구사항과 환경을 비교하고 모든 요구사항이 환경에 의해 만족되는 지를 결정함으로써 시스템에 대하여 유효화될 수 있다. 시스템은 환경에 대한 SDM 문서에서 식별된 요구사항과 시스템을 비교하고 모든 요구사항이 시스템에 의해 만족되는 지를 결정함으로써 환경에 대하여 유효화될 수 있다. 모든 요구사항이 환경 및 시스템에 의해 만족되면, 설계자 또는 개발자는 시스템이 환경 내에 배치되거나 환경 내에서 운영될 수 있는 것으로 인식한다. 그러나, 모든 요구사항이 환경 및/또는 시스템에 의해 만족되지 않으면, 만족되지 않은 요구사항을 선택적으로 알림으로써, 설계자 또는 개발

자에게 시스템이 그 환경에서 배치되고 운영되도록 SDM 문서(204)(및 대응하여 시스템) 및/또는 환경에 어떠한 변경이 수행되어야 하는지를 알린다.

[0051] SDM 문서(204) 내에 포함된 시스템의 배치에 관한 지식은 하나 이상의 환경에서 시스템이 배치되는 방법을 기재한다. SDM 문서(204)는 컨트롤러(206)에 이용가능하게 작성되고, 컨트롤러는 배치 모듈(208)과 관리 모듈(210)을 포함한다. 소정의 실시예에서, 시스템을 설치하기 위하여 필요한 시스템의 모든 파일(예를 들어, 바이너리, 데이터, 라이브러리 등) 뿐만 아니라 SDM 문서(204)는 SDU(System Definition Unit)로서 지칭되는 단일 컨테이너(예를 들어, 단일 파일)로 함께 패키징된다. 컨트롤러(206)는 도 1의 컴퓨팅 장치(102) 중 하나 이상일 수 있다. 예를 들어, 도 1의 단일 장치(102)는 특정 데이터 센터를 위한 컨트롤러일 수 있거나 또는 대안으로 컨트롤러 책임(controller resposibilities)이 다수의 장치(102)에 걸쳐 분산될 수 있다.

[0052] 배치 모듈(208)은 환경(들) 내에 시스템을 배치하는 데 사용되는 서비스를 포함한다. 도 2에서, 시스템이 그 내부에 (또는 그 위에) 배치되는 환경은 하나 이상의 타겟 장치(212)이다. 시스템은 또한 컨트롤러(206)에 배치될 수 있다. 배치 모듈(208)의 이들 서비스는 환경에 하나 이상의 시스템을 설치하거나 배치하기 위하여 호출되거나 인보크(invoke)될 수 있는 하나 이상의 기능을 포함한다.

[0053] 다른 환경에서의 배치를 위한 다른 지식은 SDM 문서(204)에 포함될 수 있다. 이 배치 지식은, 환경내에 수행되어야 하는 임의의 변경(예를 들어, 생성되어야 하는 시스템 레지스트리, 폴더, 디렉토리, 또는 파일에 대한 변경, 특정 값으로 설정되어야 하는 컴퓨팅 환경의 다른 세팅 또는 구성 파라미터 등), 환경 내의 컴퓨팅 장치(들)에 복사될 필요가 있는 어떤 파일(예를 들어, 프로그램 및/또는 데이터 파일) 및 이들 파일 상에서 수행되어야 하는 임의의 동작(예를 들어, 임의의 파일은 압축해제 및/또는 암호해독되어야 할 필요가 있을 수 있다)을 기술한다. 많은 구현예에서, SDM 문서(204) 내의 배치 지식은 예를 들어 시스템에 대한 일반적인 셋업 또는 설치 프로그램에서 현재 발견되는 것과 유사한 정보를 포함한다.

[0054] 배치 프로세스 동안, 컨트롤러(206)는 배치 내에 포함된 소프트웨어 및 하드웨어 리소스들의 레코드 또는 저장뿐만 아니라 그들 간의 관계를 발생시킨다. 이들 레코드 또는 저장은 후에 관리 페이즈 동안 컨트롤러에 의해 사용될 수 있다.

[0055] 관리 모듈(210)은 시스템이 일단 환경(들)에 설치되면 그 시스템을 관리하는 데 사용되는 서비스를 포함한다. 관리 모듈(210)의 이들 서비스는 환경 내의 시스템을 관리하기 위하여 호출되거나 인보크될 수 있는 하나 이상의 기능을 포함한다. SDM 문서(204) 내에 포함된 시스템의 관리에 관한 지식은 시스템이 하나 이상의 환경에서 관리되는 방법을 기재한다.

[0056] 다른 환경에서 시스템을 관리하는 다른 지식은 SDM 문서(204) 내에 포함될 수 있다. 관리 지식은 시스템의 관리 또는 작동에 사용되는 임의의 지식을 포함한다. 관리는 예를 들어 구성(및 선택적으로 후속하는 재구성), 패칭(patching) 및 업그레이드, 유지 태스크(예를 들어, 백업), 건강 또는 성능 모니터링 등을 포함한다.

[0057] 배치되는 시스템에 대한 변경은 관리 모듈(210)을 통해 수행된다. 관리 모듈(210)의 서비스들은 환경에 배치되는 하나 이상의 시스템에 변경을 수행하기 위하여 호출되거나 인보크될 수 있는 하나 이상의 기능을 포함한다. 관리 모듈(210)을 통한 이러한 변경을 수행함으로써, 몇개의 이점이 실현될 수 있다. 이러한 이점 중의 하나는 컨트롤러(206)가 수행된 변경의 레코드를 유지할 수 있다는 것이다. 컨트롤러(206)는 시스템을 위한 SDM 문서(204)의 복사본을 유지하고 시스템에 수행되는 임의의 변경을 SDM 문서(204)에 레코딩할 수 있다. 다른 방법으로, 컨트롤러(206)는 시스템에 수행되는 변경에 대한 별도의 레코드를 유지할 수 있다.

[0058] 컨트롤러(206)에 의해 유지되는 이 변경의 레코드는 시스템 및/또는 환경에 발생하는 문제점을 해결하거나, 또는 하드웨어 불량에 의해 시스템을 재설치(시스템이 재설치되어 불량 발생시와 동일한 파라미터/세팅으로 운영되도록 복귀될 수 있게 함)해야만 할 때 등의 후속 동작을 간략화할 수 있다. 이러한 변경이 컨트롤러(206)를 통해 수행되게 하고, 컨트롤러(206)가 레코드를 유지하게 함으로써, 일부 사람의 실수가 환경으로부터 제거될 수 있다(예를 들어, 변경을 수행하는 관리자가 그 변경을 장부에 기입할 것으로 예상되지만 그렇게 하는 것을 잊어버리면, 그 변경에 관한 레코드는 존재하지 않을 것이다. - 이 문제는 컨트롤러(206)가 레코드를 유지하게 함으로써 해결된다).

[0059] 또한, 컨트롤러(206)를 통해 시스템을 배치할 뿐만 아니라 컨트롤러(206)를 통해 시스템의 변경을 수행함으로써, 컨트롤러(206)는 환경, 환경에 배치되는 시스템, 그들 사이의 상호작용에 관한 지식의 저장소로서의 역할을 할 수 있다. 환경 및/또는 그 환경에 배치되는 시스템에 관한 지식은 컨트롤러(206)로부터 용이하게 얻을 수 있다. 이 지식은 그 환경 내의 제어된 장치가 중앙 컨트롤러(206)에 저장된 상태를 반영하는 것을

확인함으로써 제어된 환경의 일관성을 확보하는 데 사용될 수 있다.

[0060] 임의의 상황 변경이 시스템 및/또는 환경에 수행될 수 있지만, 컨트롤러(206)를 통해 수행되는 것이 아님을 주의해야 한다. 예를 들어, 컴퓨팅 장치는 갑작스럽게 오프되거나 고장날 수 있다. 이들 상황에서, 컨트롤러(206) 내의 변경을 반영하려는 시도가 이루어진다. 이들 변경은 컨트롤러(260)에서 자동적으로 반영될 수도 있고{예를 들어, 장치 고장을 검출하고 관리 모듈(210)의 서비스를 이용하여 이러한 고장을 컨트롤러(206)에 알리는 시스템이 운영될 수 있다}, 또는 컨트롤러(206) 내에서 수동적으로 반영될 수 있다{예를 들어, 관리자는 관리 모듈(210)의 서비스를 이용하여 이러한 변경을 컨트롤러(206)에 알릴 수 있다}. 다른 방법으로, 수행된 변경이 역전(reverse)되어 환경의 시스템 및/또는 부분이 컨트롤러(206)에 의해 레코딩된 바와 같이 시스템의 소망의 상태와 일치하도록 회복시킬 수 있다.

[0061] SDM 문서(204)는 "라이브(live)" 문서로서 간주될 수 있으며, 이것은 시스템의 라이프사이클 전반에서 환경에 대한 변경 및/또는 시스템에 대한 변경에 기초하여 항상 변경될 수 있다.

[0062] SDM은 수평 및 수직 축에 걸쳐 시스템의 기능 합성(function composition)을 가능하게 한다. 수평 축을 따르는 합성은 시스템 및 서브시스템으로 수행된다. 수직 축을 따르는 합성은 "계층(layers)"으로 수행된다. 애플리케이션, 서비스, 네트워크 토폴로지(network topologies) 및 하드웨어는 분산형 시스템에서 역할을 수행하지만, 통상 독립적으로 정의되고, 다른 팀 또는 조직에 의해 소유된다. 계층화는 호스트 상의 제한의 세트를 정의하는 컴포넌트에 의해 성취되며, 그 역도 성립한다.

[0063] 도 3은 계층화된 세팅을 나타낸다. 4개의 층, 즉, 층(302), 층(304), 층(306) 및 층(308)이 도 3에 도시된다. 도 3에는 4개의 층이 도시되었지만, 층의 실제 수는 변경될 수 있으며 4개보다 많을 수도 있고 적을 수도 있다. 또한, 다른 층의 콘텐츠는 다른 실시예에서 변경될 수 있다. 도 3에 도시된 바와 같이, 상이한 층들은 각기 다른 층들의 상부 및/또는 하부에 놓인다{예를 들어, 층(306)은 층(304)의 위에 놓이지만 층(308)의 아래에 놓인다}.

[0064] 층 내의 상이한 시스템들 및 서브시스템들은 서로 상호작용할 수 있고 다른 층의 시스템 및 서브시스템과 상호작용할 수 있다. 예를 들어, 층(308) 내의 서브시스템(310)은 층(308) 내의 서브시스템(312)과 상호작용할 수 있고 층(306) 내의 서브시스템(314)과도 상호작용할 수 있다. 또한, 각 층은 후속하는 상위 층에 대한 환경으로서 간주될 수 있다. 예를 들어, 층(306)은 층(308) 내의 시스템 및 서브시스템에 대한 환경인 한편, 층(304)은 층(306) 내의 시스템 및 서스시스템에 대한 환경이다. 층(302, 304, 306, 308)의 각각은 그 고유의 관련된 SDM 문서를 갖는다.

[0065] 상이한 층(302, 304, 306, 308)들은 상이한 콘텐츠를 나타낼 수 있다. 소정의 실시예에서, 층(302)은 하드웨어 층이고, 층(304)은 네트워크 토폴로지 및 오퍼레이팅 시스템 층이고, 층(306)은 애플리케이션 호스트 층이고, 층(308)은 애플리케이션층이다. 하드웨어 층은 계층화된 시스템이 설치된 물리적 장치(예를 들어, 컴퓨팅 장치)를 나타낸다{예를 들어, 도 1의 장치(102)}. 네트워크 토폴로지 및 오퍼레이팅 시스템 층은 컴퓨팅 장치의 네트워크 토폴로지 및 이들 컴퓨팅 장치 상에 설치되는 오퍼레이팅 시스템을 나타낸다{예를 들어, 도 1의 네트워크 세팅(100)}. 애플리케이션 호스트 층은 다른 애플리케이션을 호스팅할 수 있는 컴퓨팅 장치상에 설치된 애플리케이션을 나타낸다(예를 들어, SQL 서버, IIS 등). 애플리케이션 층은 다른 애플리케이션을 호스팅하지 않는 컴퓨팅 장치 상에 설치된 애플리케이션을 나타낸다(예를 들어, 게임 등의 엔터테인먼트 애플리케이션, 워드 프로세서 등의 생산성 애플리케이션, 전자 사전 등의 참조 애플리케이션, 웹 서비스 또는 금융 분석에 사용될 수 있는 분산형 애플리케이션 등).

[0066] 도 4는 시스템의 전체 라이프사이클에 걸쳐 SDM을 사용하는 프로세스(400)의 일례를 나타내는 플로우챠트이다. 도 4의 프로세스(400)의 다양한 액션은 소프트웨어, 펌웨어, 하드웨어 또는 그 조합으로 구현될 수 있다.

[0067] 초기에, 시스템은 SDM에 기초하여 설계된다(액션 402). 시스템은, 그 시스템이 환경 내에 배치되고 운영되기 위하여 그 환경이 만족시켜야 하는 요구사항, 및 시스템의 배치 및 관리에 사용되는 추가의 지식을 포함하도록 설계된다. 이 지식은 시스템과 관련된 SDM 문서 내에 포함된다. 일단 설계되면, 시스템은 SDM을 사용하여 선택적으로 유효화될 수 있다(액션 404). 이 유효화는 설계자 또는 개발자로 하여금 시스템이 유효화되는 환경에 배치되고 운영될 수 있다는 것을 검증하도록 한다. 상술한 바와 같이, 시스템만이 자신이 배치될 환경에 대해서 유효화되는 것이 아니라, 환경도 시스템에 대해서 유효화된다. 특정 환경에 대한 유효화가 실패하면, 시스템이 그 환경에서 운영될 수 있도록 시스템을 변경하기 위한 추가의 설계 단계가 수행될 수 있다 (또는 환경을 변경하기 위한 또다른 단계가 수행될 수 있다).

[0068] 일단 유효화되면, 시스템은 SDM을 사용하여 배치될 수 있다(액션 406). 환경에 시스템을 배치할 때, 시스템은 그 환경에서 후속으로 운영될 수 있도록 그 환경에 설치된다. 시스템을 설치하는 데 사용되는 지식은 시스템과 관련된 SDM 문서 내에 포함된다. 일단 배치되면, 시스템은 SDM을 사용하여 모니터링 및/또는 관리된다(액션 408). 시스템과 관련된 SDM 문서 내의 지식은 시스템이 모니터링 및/또는 관리되는 방법을 식별하고, 시스템은 이 지식에 따라 그 환경 내에서 모니터링 및/또는 관리된다.

[0069] 도 4의 프로세스(400)에는 환경인 시스템은 물론, 애플리케이션인 시스템이 사용될 수 있다. 예를 들어, 애플리케이션은 그 환경의 SDM에 대하여 유효화될 수 있다{예를 들어, 도 3의 층(308) 내의 애플리케이션은 도 3의 층(306)의 환경에 대하여 유효화된다}. 또다른 예에 의해, 오퍼레이터 또는 시스템 기획자는 환경들{예를 들어, 도 3의 층(306) 또는 층(304)}을 설계할 수 있고, 이들 환경이 배치될 환경들{예를 들어, 각각 층(304 및 302)}에 대하여 이들 환경을 유효화한다.

[0070] 시스템 및/또는 환경에 대한 제한은 런타임 동안(시스템이 모니터링 및/또는 관리되는 동안) 사용되어 런타임 동안 시스템 및/또는 환경에 대한 변경을 유효화한다. 이러한 런타임 유효화는 예를 들어 환경의 오퍼레이터가 환경에 대한 변경이 운영 시스템에 어떻게 영향을 주는 지에 대하여 결정하도록 하거나, 시스템 설계자가 시스템에 대한 변경이 그 환경에서의 시스템의 운영에 어떻게 영향을 주는 지에 대하여 결정하도록 한다.

[0071] 후술하는 바와 같이, 런타임에 대하여 플로우 및 세팅 플로우에 대한 참조가 작성된다. 플로우는 분산형 시스템의 부분들간의 구성 정보를 전달하는 데 사용된다(예를 들어, 개발자가 하나의 장소에 구성 정보를 지정하도록 하거나 오퍼레이터에게 단일 엔트리만을 제공하도록 한다). 플로우는 또한 시스템의 부분들간의 데이터를 세팅하는 플로우가 뒤따르도록 함으로써 구성에 대한 변경의 충격(impact)을 결정하는 데 사용된다.

[0072] 도 5는 SDM 런타임을 사용하는 아키텍처(500)의 일례를 나타낸다. 아키텍처(500)는 SDM 런타임(510)을 사용하는 도 2의 아키텍처(200) 뿐만 아니라 섹션 "SDM 구현의 예"에서 후술하는 SDM의 구현예이다. SDM 런타임(510)은, SDM 파일을 수락 및 유효화하고, SDU(System Definition Units - 하나이상의 SDM 파일 및 그들의 관련 파일의 패키지임)를 로딩하고, SDM 변경 요구를 생성 및 실행하고 SDM 기반 시스템을 타겟 환경에 배치하는 컴포넌트 및 프로세스의 세트를 포함한다. 런타임 기능은, SDM을 사용하여 기재된 시스템이 정의되고 유효화되고 컴퓨팅 장치의 세트에 배치되고 관리되도록 한다.

[0073] 섹션 "SDM 구현의 예"에서 후술하는 SDM은 분산형 시스템(모델링된 시스템) 내의 컴포넌트들의 구성, 상호작용 및 변경의 설명을 지원하도록 설계된다. SDM은 객체 관계 모델에 기초한다. "정의"는 시스템 내에 존재하는 엔티티(entities)를 기술하고 "관계"는 다양한 엔티티들간의 링크를 식별한다. 정의와 관계는 또한 SDM에 관련된 시멘틱(semantic) 정보를 캡처하도록 정의된다. 특히, 정의는 컴포넌트, 엔드포인트 및 리소스로 분할된다. 관계는 접속(또한 통신으로 지칭됨), 포함(containment), 호스팅, 델리게이션(delegation) 및 참조로 분할된다. 정의와 관계에 대한 더 상세한 설명은 이하에서 제공된다.

[0074] SDM은, 시스템 부분의 공통 카테고리화를 제공하고 광범위한 시스템에 대한 지원 툴을 제공하고 설계시에 정의 검사를 위한 기본을 제공하는 "추상 정의(abstract definitions)"를 포함한다. 추상 정의의 세트는 서비스 설계에 대한 포괄적인 기본을 제공한다. "구체적인 정의(concrete definitions)"는 실제 시스템 또는 데이터 센터 설계의 부분들을 나타낸다. 구체적인 정의는, 추상 정의를 선택하고 구체적인 정의의 멤버 및 그 특성에 대한 세팅 값을 정의하는 구현을 제공함으로써 생성된다. 분산형 애플리케이션은 이들 구체적인 정의의 수집을 사용하여 생성된다.

[0075] SDM은 또한 관계의 인스턴스가 참여할 수 있는 허용된 관계의 세트에 기초하여 한정(restriction)을 모델링하는 "제한(constraint)"을 포함한다. 제한은 관계에 포함되는 객체의 구성에 의존하는 요구사항을 설명하는 데 유용하다. 예를 들어, 제한은 통신 프로토콜의 각 엔드 상의 참여자가 호환가능한 보안 세팅을 사용하는지를 결정하는 데 사용될 수 있다.

[0076] 타겟 시스템에 대한 변경을 달성하기 위하여, SDM은 "변경 요구(Change Request)" 또는 CR이라 지칭되는 요구된 변경의 선언적 설명(declarative description)을 사용한다. SDM은, "SDM 실행 모델"의 부분으로서 변경 요구를 확장하고 유효화하고 실행하는 데 사용되는 프로세스를 정의한다.

[0077] "인스턴스 공간"은 관리되는 애플리케이션의 원하는 상태 및 현재의 상태 둘다를 모두 캡처한다. 인스턴스 공간의 변경은 추적(track)되며 변경을 개시한 변경 요구와 관련된다. 인스턴스 공간은 SDM 런타임에 저장되고 모델링된 시스템의 현재 상태를 반영한다. 런타임은, 생성된 인스턴스와 이들 인스턴스간의 관계의 완전한 레코드를 포함한다. 각각의 인스턴스는 각각의 버전이 변경 요구에 링크된 관련 버전 히스토리를 갖는다. 새로

운 인스턴스를 생성하는 프로세스는 변경 요구에 의해 개시된다. 변경 요구는 기존의 인스턴스의 특정 멤버와 관련된 정의와 관계에 대한 생성, 갱신 및 삭제 요구의 세트를 정의한다.

[0078] 다음은 도 5의 컴포넌트들이 함께 작동하는 방법에 대하여 간략한 기능을 설명한다. 오퍼레이터 또는 관리자는 데이터 센터의 토폴로지와 같은 애플리케이션이 배치될 수 있는 환경을 설명할 수 있다. 오퍼레이터 또는 관리자는 환경을 설명하는 SDM 파일을 생성하며, 이 파일은 "논리적 인프라스트럭쳐"(LIM; 502)로서 지칭되거나 데이터 센터 설명 또는 데이터 센터 모델로서 지칭된다. 이 SDM 파일은 워싱톤주 레드몬드의 마이크로소프트 코포레이션으로부터 이용가능한 비주얼 스튜디오 개발 시스템과 같은 다양한 개발 시스템 중의 임의의 것을 사용하여 발생될 수 있다.

[0079] 또한, 애플리케이션 개발자는 비주얼 스튜디오 개발 시스템과 같은 다양한 개발 시스템 중 임의의 것을 사용하여 그들의 애플리케이션을 설계 및 개발할 수 있다. 개발자가 애플리케이션의 컴포넌트와 이들 컴포넌트가 서로 어떻게 관련되는지를 정의함에 따라, 개발자는 데이터센터 설명(502)에 대한 애플리케이션의 설명을 유효화할 수 있다. 이것은 또한 "설계 시간 유효화(Design Time Validation)"로서 지칭된다.

[0080] 애플리케이션이 완성되면, 개발자는 SDM에 설명을 저장하고, 애플리케이션이 SDU(504)로서 배치되기 위해 패키징될 것을 요구한다. SDU는 애플리케이션 SDM 뿐만 아니라 애플리케이션 바이너리 및 애플리케이션을 설치하는데 사용되는 다른 참조 파일을 포함한다.

[0081] LIM(502) 및 SDU(504)는 배치를 위하여 컨트롤러 장치(520)의 배치 툴(506)에 공급된다. 배치 툴(506)은 사용자 인터페이스(UI)를 포함하여 오퍼레이터가 소망의 SDU(504)를 로드할 수 있게 한다. 배치 툴(506)은 생성 CR 모듈(530)로 작업하여 SDU(504)의 SDM 내의 정보에 따라 SDU(504)와 관련된 애플리케이션을 설치한다. 또한, SDU(504)로부터의 SDM 정의와 인스턴스는 SDM 런타임(510)의 저장부(508)에 저장된다. SDU는 SDU 관리 모듈(540)에 의해 SDM 런타임(510)에서 관리되고, SDU 관리 모듈은 SDU의 적절한 부분들이 런타임(510) 및 타겟(들)(522)의 다른 컴포넌트들에 이용가능하도록 한다.

[0082] 오퍼레이터는 또한 애플리케이션이 배치되는 타겟(522; 예를 들어, 타겟 컴퓨팅 장치) 상에서 그가 취하기를 원하는 액션이 무엇인지를 지정할 수 있다. 오퍼레이터는 여기에서 변경 요구(CR)로서 지칭되는 배치 파일을 통해 배치를 수행할 수 있다. CR은 하나 이상의 엔진(512, 514, 516, 518)을 통해 운영된다. 일반적으로 확장 CR 엔진(512)은 CR을 확장하여 관련된 모든 컴포넌트 뿐만 아니라 그들 접속 및 액션을 식별하고, 플로우 값 엔진(514)은 컴포넌트들에 대한 값(접속 스트링 등)을 플로우(flow)하고, 제한 검사 엔진(516)은 환경 및 애플리케이션 간의 제한을 검사하고, 액션 순서 엔진(518)은 CR에 대해 필요한 모든 액션에 대한 순서를 지정한다.

[0083] 시스템에 대한 변경(애플리케이션의 배치를 포함) 또는 모델의 유효화를 개시하기 위하여, 오퍼레이터 또는 프로세스는 CR을 제출한다. CR은 오퍼레이터가 런타임(510) 내의 인스턴스를 통해 수행하기를 원하는 액션의 세트를 포함한다. 이들 액션은 예를 들어 생성 액션, 갱신 액션, 및/또는 삭제 액션일 수 있다.

[0084] 사용자 또는 오퍼레이터가 변경 요구를 개시하는 것에 더하여, 후술하는 바와 같이, 확장 프로세스의 부분으로서 발생된 확장/자동 발생된 변경 요구가 있을 수 있다. 그들의 소스에 관계없이, 일단 완전히 확장되고 검사되면, 변경 요구는 타겟(522)에 타겟 인스턴스를 탐색, 설치, 분리 및 변경하는 등의 액션을 송신함으로써 실행된다.

[0085] CR은 그룹으로서 완료하거나 실패한 작은 세트의 액션으로서 취급된다. 이것은 예를 들어 제한 검사 엔진(516)이 유효성을 검사할때 모든 액션을 고려하도록 한다.

[0086] 설계 시간 유효화시에, CR은 SDM 컴파일러(528)에 의해 생성될 것이며 SDM 파일 내의 각각의 SDM 컴포넌트의 하나 또는 최소를 포함할 것이다. 이러한 생성 인스턴스 코맨드의 CR은 확장 엔진(512), 플로우 값 엔진(514), 및 제한 검사 엔진(516)을 통해 플로우한다. 이들 세개의 페이즈에서 탐색된 에러는 사용자가 사용하고 있는 개발 시스템을 통해 그 사용자에게 리턴될 것이다.

[0087] 배치에 있어서, 오퍼레이터는 개발 툴(506)에 의해 부여된 UI를 사용하여 CR을 생성할 것이다. CR은 SDM 런타임(510) 내의 모든 엔진(512, 514, 516 및 518)을 통해 플로우하고, 적절한 액션 및 정보가 CR 모듈(532)에 의해 요구가 실행되는(예를 들어 애플리케이션이 설치되는) 적절한 타겟(들)(522)에 송신될 것이다. 특정 설치를 위한 적절한 타겟(들)(522)은 통상 애플리케이션이 설치되는 타겟(들)이다.

[0088] 정의 분석 페이즈에서, CR을 처리하기 시작하면, 생성 CR 모듈(530)은 변경 요구에서 참조되는 모든 정의와 멤

버를 분석한다. 변경 요구는, 이들이 런타임(510)에 의해 이미 로딩된 것으로 추정할 것이며, 그들이 존재하지 않으면 생성 CR 모듈(530)이 로드/컴파일 액션을 개시한다. 생성 CR 모듈(530)은 또한 기존의 인스턴스 및 변경 요구 내의 생성 액션에 의해 정의된 인스턴스에 대한 참조가 분석되는 경로 분석 페이즈를 실시한다.

[0089] 확장 엔진(512)에 의해 수행되는 확장은, 변경 요구가 주어지면, 요구를 실행하는 데 필요한 나머지 모든 액션이 파퓰레이트되는 프로세스이다. 일반적으로, 이들 액션은 정의 및 관계 인스턴스에 대한 구성 및 해체 액션이다. 오퍼레이터는 인스턴스의 구성 또는 해체에 필요한 모든 액션에 대한 세부사항을 제공할 수 있고, 다르게는 프로세서의 부분들이 자동화될 수 있다: 예를 들어, 오퍼레이터는 멤버(예를 들어, byReference 멤버)에 관한 액션을 식별함으로써 그가 원하는 변경에 대한 키 정보를 제공하고, 액션의 나머지는 네스트된 멤버(예를 들어, byReference 및 byValue 멤버) 및 관계에 제공된다. 다른 예에 의해, 자동화된 확장은, 또한 가용한 리소스를 갖는 장치를 선택하고 요구하는 데이터에 근접하여 애플리케이션을 위치시키는 것 등에 기초하여 배치 결정을 할 수 있는 외부 리소스 매니저로 지칭될 수 있다.

[0090] 확장 엔진(512)은 또한 "자동 기록"을 수행한다. 자동 기록 동안, 엔진(512)은 SDM 내에 지정된 합성 컴포넌트 및 컴포넌트의 스케일 불변 그룹화를 분석하고 컴포넌트가 요구된 레벨로 스케일링될 때 어떻게 그룹화되고 상호 접속될지를 결정한다.

[0091] 확장 엔진(512)은 또한 값 멤버 확장, 참조 멤버 확장 및 관계 확장을 수행한다.

[0092] 값 멤버 확장은 모든 비참조(non-reference) 정의 멤버의 식별을 지칭한다. 이들 멤버의 카디널리티(cardinality)가 인식되고, 모든 요구된 파라미터가 공지되므로, 각각의 멤버에 대하여, 생성 요구는 그 부모(parent)가 생성중인 멤버들에 대한 변경 요구에 부가된다. 변경 요구가 해체 동작을 포함하면, 해체 동작은 그들의 모든 포함 인스턴스에 대하여 부가된다.

[0093] 참조 멤버 확장은 (비참조 정의 멤버와 반대로) 참조 멤버로 지칭된다. 참조 멤버의 카디널리티는 종종 정의되지 않으며 참조 멤버는 인스턴스가 구성되도록 값을 요구하는 배치 시간 세팅을 가질 수 있다. 그래서, 참조 멤버(예를 들어, byReference 멤버)를 확장하는 프로세스는 제공하기 위한 위치에 있는 런타임보다 인스턴스에 대하여 더 많은 정보를 요구할 수 있다.

[0094] 참조 멤버 확장은, 이미 배치된 인스턴스를 찾는 데 사용되는 탐색(discovery)으로서 지칭되는 프로세스와 관련된다. 탐색은 환경의 오퍼레이터에 의해 통상 개시되는 액션이다. 예를 들면, 설치 요구 동안, 확장 엔진(512)은 인스턴스가 이미 존재하는지를 판정하고, 이미 존재하는 것으로 판정된 경우에는 어떤 것이 존재하는지를 판정하며, 존재하지 않는 것으로 판정된 경우에는 인스턴스를 생성한다. 컨트롤러(520) 상의 인스턴스 매니저(IM)(534)는 타겟 장치(522) 상의 인스턴스 매니저(526)와 통신하여 탐색 프로세스를 개시한다. 탐색 프로세스는 인스턴스에 관한 데이터를 타겟 장치(522)로부터 컨트롤러(520)로 리턴한다.

[0095] 탐색 프로세스는 구성 또는 갱신 액션의 부분으로서 참조 정의 멤버를 파퓰레이트한다. 통상, 탐색을 지원하는 객체 매니저(또한 탐색을 수행하는 인스턴스 매니저)를 갖는 참조 멤버만이 이 프로세스에 참여한다.

[0096] 새로운 인스턴스가 탐색될 때, 인스턴스 특정 키 값을 사용하여 인스턴스가 SDM 데이터베이스 내에 이미 존재하지 않는지를 검사한다. 일단 그것이 새로운 인스턴스인 것으로 알려지면, 인스턴스는 탐색된 멤버들의 정의에 따라 분류된다. 인스턴스가 멤버와 일치하지 않거나 확실하지 않게 일치되면, 멤버 참조는 블랭크 상태로 남고 인스턴스는 오프라인 및 비완료로서 마크된다.

[0097] 구성될 모든 정의 인스턴스가 공지되면, 관계 확장은 정의 인스턴스를 함께 결합하는 생성 관계 인스턴스를 참조한다. 정의 인스턴스가 해체되면, 정의 인스턴스를 참조하는 모든 관계 인스턴스는 제거된다.

[0098] 관계를 생성하기 위하여, 멤버 공간은 인스턴스들 사이에서 존재해야 하는 관계의 구성을 식별하는 데 사용된다. 정의 멤버가 1보다 큰 카디널리티를 가지면, 관계 토폴로지는 베이스 관계 정의로부터 추론된다. 예를 들어, 통신 관계에 대하여, "자동 기록"이 수행될 수 있고, 호스트 관계에 대하여, 호스팅 관계와 관련된 알고리즘에 기초하여 호스트가 얻어질 수 있다.

[0099] 플로우 스테이지에서, 플로우 값 엔진(514)은 모든 관계 인스턴스에 걸친 플로우를 평가한다. 플로우 값 엔진(514)은 변경된 임의의 파라미터 플로우에 의해 영향을 받은 인스턴스에 대한 변경 요구에 갱신 요구를 부가할 수 있다. 엔진(514)은 변경 요구의 결과로서 갱신된 세팅을 갖는 인스턴스의 세트를 결정함으로써 플로우를 평가한다. 이들의 각각에 대하여, 변경된 세팅에 의존하는 임의의 아웃고잉 세팅이 평가되고 타겟 노드는 변경된 인스턴스의 세트에 부가된다. 프로세스는 세트가 비워지거나 세트가 한 사이클을 포함할 때까지 계속된다.

[0100] 플로우 스테이지 이후에, 이중 검출의 프로세스가 수행된다. 이중 검출은 도 5에 도시된 엔진{예를 들어, 플로우 값 엔진(514) 또는 제한 검사 엔진(516)} 중의 하나 또는 도 5에 도시되지 않은 또다른 엔진에 의해 수행될 수 있다(예를 들어, 이중 검출 엔진은 SDM 런타임(510)에 포함될 수 있다). 이중 검출의 프로세스는 SDM 데이터 저장부에 이미 존재하는 인스턴스에 대하여 확장된 인스턴스를 일치시킨다. 예를 들어, 프로세스는 또다른 애플리케이션이 공유 파일을 설치했는지를 검출한다. 이미 존재하는 인스턴스가 검출되면, 몇개의 액션 중 하나가 기존의 인스턴스의 버전에 의존하여 수행될 수 있다. 설치가 실패할 수 있고; 인스턴스가 카운트된 참조일 수 있거나; 인스턴스가 업그레이드될 수 있거나; 또는 설치가 나란히 수행될 수 있다.

[0101] 제한 검사 엔진(516)은 변경 요구가 처리된 후에 모든 제한이 여전히 유효한지를 보기 위하여 모델 내의 모든 제한이 검사되는 제한 평가 페이즈를 실시한다.

[0102] 제한 검사 엔진(516)이 제한 평가 페이즈를 종료한 후, 액션의 완전한 리스트가 이용가능하다. 그래서, 액션 순서 엔진(518)은 컴포넌트들 사이의 관계를 사용하여 유효한 변경 순서를 결정한다. 많은 알고리즘 중의 임의의 것이 사용되어 이 결정을 수행한다.

[0103] 순서를 결정하는 액션 순서 엔진(518)이 완료되면, 머신 특정인 순서화된 액션의 세트의 서브세트를 분산함으로써 배치가 수행될 수 있다. 일단 머신에 의해 액션들이 순서화되고 그룹화되면, 액션 뿐만 아니라 인스턴스 정보를 갖는 SDM 런타임 저장부(508)의 필요한 부분의 복사본이 타겟 컴퓨팅 장치(522)로 송신된다. SDM은 저장 캐시(538) 내의 타겟 장치에서 일시적으로 저장된다.

[0104] 타겟 컴퓨팅 장치는 SDM 런타임(510)과 통신하는 SDM 런타임의 타겟 부분(536)을 포함한다. 타겟 컴퓨팅 장치(522)는, 또한 실행 엔진(524)을 포함하고 타겟 장치 상의 적절한 인스턴스 매니저(IM; 526)와 통신하여 생성, 갱신 및 삭제 액션 등의 타겟에 대한 변경을 수행하는 에이전트를 포함한다. 각각의 액션은 인스턴스 매니저(526)로의 호출로서 송신되고 인스턴스 매니저(526)는 상태 메시지를 리턴하고, 일부 액션에 대해서는 데이터를 리턴한다(예를 들어, 탐색을 위하여). 모든 액션이 타겟(522) 상에서 완료되면, 타겟의 에이전트는 임의의 에러 및 상태를 컨트롤러(520)로 리턴한다. 그후, 컨트롤러(510)는 이 정보를 사용하여 SDM 런타임 저장부(508)를 갱신한다.

[0105] 상술한 바와 같이, 변경은 영향을 받은 관계에 기초하여 변경 요구를 분산가능한 부분으로 분류함으로써 수행된다. 모든 부분이 완료되면(또는 하나 이상이 실패한 후) 결과는 런타임(510)에서 대조되고 요약이 오퍼레이터에 리턴된다. 실패한 경우, 모든 액션은 "롤백(rolled back)"되고, 시스템은 변경이 개시되기 전의 상태로 리턴된다.

[0106] 소정의 실시예에서, 상술한 설계 시간 유효화 동안, SDM 컴파일러(528)는 SDM 파일을 수신하고, 테스트 CR을 생성하고, SDM 런타임의 확장, 플로우 값 및 제한 검사 엔진을 통해 테스트 CR을 운영하고, 임의의 에러를 개발 시스템에 리턴한다. 이 프로세스는 개발자를 위하여 설계 시간동안 배치에 대한 SDM 유효화를 제공한다.

[0107] SDM 런타임(510) 및/또는 컨트롤러(520)로의 공중 인터페이스는 객체 모델(API) 라이브러리를 통한다. 라이브러리는 관리된 코드 객체 모델이며 다음이 수행되도록 한다.

[0108] • 런타임 내에서 SDM을 관리 - SDM 파일은 런타임으로 로딩될 수 있다. SDM은 불변이며 한번에 하나씩 로딩된다 {즉, 파일의 부분(예를 들어, SDM 파일로부터의 개별 정의, 클래스 또는 맵핑로부터의 하나)보다는 오히려 하나의 SDM 파일이 로딩될 수 있다}. SDM은 런타임으로부터 삭제될 수 있고 런타임 내의 SDM에 대한 XML 문서가 생성될 수 있다.

[0109] • 런타임에 의해 공지된 SDU를 관리.

[0110] • SDM 정의를 관리 - (런타임 내에 로딩된 SDM으로부터) SDM 요소를 탐색하고 반영. 새로운 SDM을 작성하는데 제공되는 공중 API는 없다(즉, 이것은 SDM의 불변 요소를 통한 판독 전용 객체 모델이다). 이것은 SDM, SDU, 식별정보(identity), 버전, 클래스, 정의, 결합/맵핑 및 버저닝 정책(versioning policy)을 포함한다.

[0111] • SDM 인스턴스를 관리 - 컴포넌트, 엔드포인트, 리소스 및 관계의 인스턴스를 탐색하고 반영. 인스턴스 공간에서, 각각의 인스턴스는 GUID, 안정한 경로 또는 어레이 기반 경로에 의해 식별될 수 있다. 경로는 스트링이며 상대적일 수 있다. 상대적인 경로를 포함하는 이들 식별자는 인스턴스가 변경 요구 문서 등의 문서 내에서 탐색되고 참조될 수 있도록 한다.

[0112] • 인스턴스 조작 - 생성, 토폴로지 변경, 업그레이드, 세팅 변경 및 삭제를 포함하는 SDM 인스턴스에 대한 변경을 수행. 인스턴스 변경은 갱신의 최소 유닛을 제공하는 변경 요구의 경계 내에서 수행되어, 어떠한 에러 또는 제한 위반에 의해서도 전체 요구가 실패하게 한다. 요구가 커미트(commit)되면 인스턴스가 호스트를 가져야 하므로, 인스턴스 요구는 또한 호스트로의 결합없이 인스턴스가 일시적으로 존재하도록 한다. 또한 단일 컴포넌트의 설치 또는 세팅에 영향을 주는 많은 동작이 수행되고, 설치 또는 세팅 갱신을 커미트시까지 연기시켜, 단일 갱신이 컴포넌트 상에서 발생하게 한다. SDM 모델 검사는 변경 요구 커미트 시간 이전 또는 그 시간에 수행되고 커미트는 임의의 모델 또는 제한 위반으로 실패할 것이다.

[0113] • 변경 요구를 로드 - 변경 요구는 인스턴스 공간 동작의 세트를 나타내는 문서, 예를 들어, XML 파일이다. 이 문서는 애플리케이션 인스턴스를 생성 또는 삭제하기 위한 재사용가능한 '스크립트'가 되도록 상대적인 경로를 이용할 수 있다.

[0114] • 변경 요구에 대한 탐색 및 반영 - 설치/갱신 태스크 및 모든 에러 정보의 획득, 및 요구에 의해 영향을 받은 컴포넌트의 설치/갱신의 재시도를 포함.

[0115] • 데이터베이스 내의 변경 요구로부터 변경 요구 문서 발생. 이러한 문서는 어느 정도는 이식가능(portable)하다.

[0116] • 프로그레스, 로그 또는 상태 갱신 등의 변경 요구 태스크에 대한 이벤트에 가입. 이들 이벤트 가입의 수명은 클라이언트를 로드한 프로세스의 수명에 의해 제한된다(즉, 이들은 규칙적인 CLR 이벤트이다).

[0117] SDM 런타임 엔진은 SDM 모델에 관한 증명 및 API에 의해 표면화된 기능을 수행한다. 라이브러리는 (SDM 엔티티 상에 반영하기 위하여) SDM 로드, 컴포넌트 인스턴스 생성 및 전체 SDM 획득과 같이 상당히 정밀하지 않은 호출(fairly coarse call)로 웹 서비스로서 런타임 엔진과 통신한다. 이 웹 서비스를 위한 파라미터의 다수의 포맷은 SDM 파일을 위한 동일 스키마를 갖는 XML이다. 엔진은 또한 허가(permission)에 대하여 검사를 수행할 수 있다.

[0118] 컨트롤러(520)는 인스턴스 매니저(IM)를 사용할 수 있고, 인스턴스 매니저는 모델 내의 임의의 정의 또는 관계와 관련될 수 있다. IM은 다음의 규칙 중의 하나 이상을 수행할 수 있다.

[0119] • 인스턴스의 배치 지원.

[0120] • 일단 배치되면{감사(audit)}, 인스턴스의 유효화 지원.

[0121] • 이미 배치된 인스턴스들 중에서 런타임을 통해 배치되지 않은 인스턴스를 탐색 지원.

[0122] • 세팅 값의 플로우 지원.

[0123] • 제한의 평가 지원.

[0124] • 변경 요구의 확장 지원.

[0125] • API를 통한 CLR 클래스로서 사용자에게 인스턴스 제공 지원.

[0126] 배치를 위하여, 컨트롤러(520) 상의 인스턴스 매니저(IM) 플러그인은, 클래스 호스트 관계와 관련되고, 클래스를 위한 설계 경험을 제공하고 SDU(540) 내의 관련 바이너리 및 세팅 스키마를 생성하는 개발 시스템 내에서 사용되는 플러그인으부터 분리된다. 인스턴스 매니저는, 인스턴스 매니저 인터페이스를 구현하거나 추상 클래스로부터 받은 (예를 들어 dll 어셈블리 내의) CLR 클래스로서 SDM 런타임(510)에 공급된다. 인스턴스 매니저(IM) 플러그인으로서도 지칭되는 SDM 인스턴스 매니저는 컨트롤러(520)에 다음의 기능을 제공한다.

[0127] • 파일 및 코맨드(태스크)를 발생하여 그들의 호스트 상에 컴포넌트 인스턴스를 설치, 분리 또는 재설치함 - 변경 요구가 새로운 컴포넌트 인스턴스, 컴포넌트 인스턴스의 제거, 또는 분리 또는 재설치를 요구하는 컴포넌트로의 변경을 발생시키면, SDU(204) 내에서 인스턴스, 호스트 인스턴스, 컴포넌트와 관련된 정의 및 이들 정의와 관련된 바이너리에 대한 세팅을 획득하고, 배치 엔진을 통한 수동 실행 또는 디스패치(dispatch)를 위해 준비된 타겟 서버 상에 설치 또는 분리를 수행하는 데에 필요한 파일 및 코맨드를 생성하는 것을 인스턴스 매니저이다.

[0128] • 파일 및 코맨드(예를 들어, 태스크)를 발생하여 그 세팅 변경시 또는 그 엔드포인트 중의 하나로부터의 관찰이 변경될 때 (예를 들어, 통신 관계 토폴로지 변경 또는 가시적인 엔드 포인트가 세팅 변경을 갖는 것에 의해) 컴포넌트 인스턴스를 갱신.

[0129] • 컴포넌트 인스턴스의 엔드포인트 상의 가시적인 엔드포인트 인스턴스를 컴포넌트 인스턴스 상의 세팅으로 맵핑 - SDM에서, 컴포넌트 인스턴스는, 일부의 통신 관계 토폴로지의 결과로서, 다른 엔드 포인트 인스턴스를 볼 수 있는 엔드포인트 인스턴스를 갖는다. 다른 엔드포인트 인스턴스의 세부사항은 컴포넌트 인스턴스가 런타임에서 페칭(fetch)되어 결합할 수 있도록 하는 세팅으로 맵핑된다. 예를 들어, 웹 사이트는 데이터베이스 클라이언트 엔드포인트 인스턴스를 가질 수 있고, 따라서 통신 관계가 데이터베이스로서 확립될 수 있다. 정확하게 확립되면, 그 데이터베이스 클라이언트 엔드포인트는 단일 데이터베이스 서버 엔드포인트 인스턴스 및 그 서버 엔드포인트 상의 세팅을 관찰할 수 있다. 이 정보는 인스턴스 매니저에 의해 사용되어 클라이언트 엔드포인트의 이름하에 구성 파일 내의 서버에 대한 접속 스트링을 배치한다. 최종 결과는 코드가 그 구성 세팅으로부터 데이터베이스에 대한 접속 스트링을 간단히 판독하는 것이다.

[0130] • 파일 및 코맨드(태스크)를 발생하여 컴포넌트 인스턴스를 편집 - 편집은 존재 및 정확한 세팅을 확인한다. 이것은 또한 호스트 인스턴스 세팅에 적용될 수 있다.

[0131] • 임의의 태스크에 대하여 상태를 보고함 - IM은 캡처된 부분 또는 완전한 출력을 변형하고 그 태스크의 상태를 성공, 실패, 불완전 및 불완전에 대한 선택적 제공 프로그레스(% 또는 최종 응답), 실패에 대한 세부사항(에러 메시지) 및 임의의 상태에 대한 인간 판독가능 로그로서 제공한다. 인스턴스 매니저로 되돌아가서 태스크의 출력을 해석함으로써, 인스턴스 매니저는 인간 판독가능한 채로 유지되면서 진단을 위한 충분한 로깅을 생성해야 하도록 시도하기 보다는 그 태스크 로그 구조화 정보(예를 들어 XML 또는 심지어 SOAP)를 자유롭게 갖는다.

[0132] • 인스턴스 매니저는 또한 호스트와 그들의 게스트들간의 제한 검사를 수행하는 코드를 포함할 수 있다. 설치자는 예를 들어, XML, XPath 및 XQuery에 기초한 공통 제한 언어를 사용할 수 있다.

[0133] **SDM 구현예**

[0134] 다음의 설명은 SDM의 요소를 정의하는 스키마의 실시예를 기재한다.

[0135] **1. 정의**

[0136]

용어	정의
변경 요구	변경의 세트를 모델링된 시스템으로 설명하는 선언적 문서
완전히 인가된 변경 요구	모델 평가 스테이지를 통과하고 현재 타겟 시스템에 대하여 실행될 준비가 된 변경 요구
추상 타입	모델링된 시스템 객체에 대하여 작동하기 위하여 필요한 세팅을 정의하는 데 사용되는 타입
구체적 타입	멤버 타입과 관계에 대한 정의를 포함하는 모델링된 시스템 객체의 재사용가능한 정의
관계	모델링된 시스템 요소들간의 상호작용을 설명하는 데 사용되는 sdm 객체
시스템 정의 모델(SDM) 문서	추상 객체, 구체적 타입 및 관계에 대한 정의를 포함하는 xml 문서
소프트웨어 분산 유닛(SDU)	타입을 SDM 관리 시스템에 배치하는 데 사용되는 관련 바이너리 정보(파일) 및 SDM 문서의 세트의 조합
SDM 층	층은 그 층 내에 모델링된 객체에 지정된 추상 객체의 세트이다. 예를 들어, 애플리케이션 층 타입은 웹 애플리케이션 및 데이터베이스를 포함할 수 있지만, 오퍼레이팅 시스템 층은 파일 시스템 및 네트워크 장치를 위한 타입을 포함할 수 있다. 어떤 타입은 층에 할당되지 않고 대신에 층의 범위에 걸쳐 사용될 수 있다.
SDM 인스턴스 공간	모델링된 시스템을 나타내는 구체적 타입과 관계 인스턴스의 세트

[0137] **2. 아키텍처 개요**

[0138] 시스템 정의 모델(SDM)은 분산형 시스템(모델링된 시스템) 내의 컴포넌트들의 구성, 상호 작용 및 변경의 설명을 지원하기 위하여 설계된다.

[0139] SDM은 객체 관계 모델에 기초한다. 우리는 객체를 사용하여 시스템 내에 존재하는 엔티티와 관계를 설명하여 그들간의 링크를 식별한다. SDM은 또한 객체 및 관계를 구별하고 SDM에 중요한 시멘틱을 캡쳐한다. 특히, 우리는 객체를 시스템, 엔드포인트 및 리소스로 분할하고, 관계를 통신, 포함(containment), 호스팅, 델리게이션(delegation) 및 참조로 분할한다.

[0140] 우리는 추상 정의를 사용하여 광범위한 애플리케이션을 위한 툴 지원을 허용하고 설계 시간에 검사하는 타입에 대한 기본을 제공하는 시스템 부분의 공통 카테고리화를 제공한다. 우리는 추상 정의의 세트를 예상하여 시스템 설계에 대한 포괄적인 기본을 제공하고 그들이 시간에 따라 천천히 변경될 것을 예상한다.

[0141] 우리는 실제 애플리케이션의 부분 또는 데이터센터 설계를 나타내는 구체적 객체 정의를 작성한다. 우리는 추상 객체 정의를 취하고 구체적 타입의 멤버 및 그 특성에 대한 세팅 값을 정의하는 구현을 제공한다. 그후, 우리는 이들 정의의 수집으로부터 시스템을 작성한다.

[0142] 제한은 인스턴스가 참여할 수 있는 관계의 허용된 세트에 관한 한정을 모델링하는 데 사용된다. 우리는 제한을 사용하여 관계에 포함된 객체의 구성에 의존하는 미세한 요구사항(fine-grained requirements)을 캡쳐한다. 예를 들어, 제한은 통신 프로토콜의 각각의 엔드 상의 참여자가 호환가능한 보안 세팅을 사용하는 것을 유효화하는 데 사용될 수 있다.

[0143] 타겟 시스템 상의 변경을 달성하기 위하여, SDM은 변경 요구라 불리우는 요구된 변경의 선언적 설명을 사용한다. SDM은 SDM 실행 모델의 부분으로서 변경 요구를 확장, 유효화 및 실행하는 데 사용되는 프로세스를 정의한다.

[0144] 인스턴스 공간은 관리된 애플리케이션의 원하는 및 현재의 상태 모두를 캡쳐한다. 우리는 인스턴스 공간의 변경을 추적하고 이들을 변경을 개시하는 변경 요구와 관련시킨다.

[0145] 다음의 uml 다이어그램은 sdm 모델 내의 객체들 사이의 광대한 상호작용을 캡쳐한다. 간략화를 위하여, 이들 상호작용 중의 일부는 베이스 타입들 사이에 정의되었으며, 실제 상호 작용은 도출된 타입(derived type)들 사이에 존재하고 따라서 더 특수화된다. 예를 들어, 통신 관계는 추상 엔드포인트 정의만을 참조할 수 있다.

[0146] Sdm 문서는 문서를 설명하는 정보, 문서 내의 정의에 대한 매니저, 다른 문서를 참조하는 임포트 명령문(import statement) 및 정의 세트를 포함한다.

[0147] 도 6은 문서의 일례를 나타낸다.

[0148] 모든 sdm 정의는 공통 기본 정의로부터 도출되고 도 7에 도시된 바와 같이 멤버를 포함할 수 있다. 정의와 멤버 간의 관계는 다음의 도면에 도시된 것보다 더 복잡할 수 있다.

[0149] 멤버는 도 8에 도시된 것을 참조한 정의의 종류에 의해 분할된다.

[0150] 세팅 선언은 세팅 정의를 참조한다. 세팅 값과 값 리스트는 도 9에 도시된 바와 같이 세팅을 위한 값을 제공한다.

[0151] **2.1 SDM 애플리케이션의 라이프사이클**

[0152] 소정의 실시예에 따른 SDM 애플리케이션의 라이프사이클의 일례가 도 10에 도시된다.

[0153] 애플리케이션은 비쥬얼 스튜디오 환경 내에서 설계되고 구현된다(블록 1002). 개발자는 컴포넌트들을 구현한 후 그들을 복합 컴포넌트들 내에서 결합한다. 애플리케이션은 SDM 파일 내에 설명된다. 그들의 애플리케이션이 특정 데이터센터 내에 배치되는 것을 검증하기 위하여, 개발자는 그들의 애플리케이션을 SDM 파일 내에 설명된 데이터센터의 표시에 결합할 것이다(블록 1004). 이 표시는 그들의 애플리케이션 컴포넌트들의 호스트들에 대한 정의와 그들의 애플리케이션의 구성에 대한 제한을 포함할 것이다. 결합이 실패하면, 개발자는 그들의 애플리케이션 설계를 변경할 수 있다.

[0154] 개발자가 일단 그들의 애플리케이션으로 만족하면, 애플리케이션과 관련된 강력한 이름과 버전으로 그 애플리케이션을 서명하여 공표한다(블록 1006). 애플리케이션의 공표된 형태는 소프트웨어 분산 유닛(SDU)이라 불리운다. 오퍼레이터는 개발자로부터 SDU를 얻어 애플리케이션을 SDM 런타임으로 로딩한다(블록 1008). 애플리케이션을 로딩하는 프로세스에서, 오퍼레이터는 그들이 애플리케이션을 결합하기를 원하는 데이터센터의 모델을 선택한다. 오퍼레이터가 애플리케이션을 배치할 것을 선택하면, 그들은 배치 시간 파라미터를 애플리케이션에 공급하고, 그들은 애플리케이션의 스케일을 결정한다(블록 1010). 이것은 변경 요구를 사용하여 수행된다.

[0155] 일단 애플리케이션이 배치되면, 오퍼레이터는 런타임과 상호작용하여 애플리케이션의 구성과 애플리케이션의 각 부분에 대한 세팅을 결정할 수 있다(블록 1012). 런타임은 또한 애플리케이션의 실제 구성이 런타임 내에 레코딩된 원하는 구성과 일치하는지를 검증할 수 있다. 오퍼레이터는 변경 요구를 제출함으로써 배치된 애플리케이션을 제거할 수 있다(블록 1014). 오퍼레이터는 또한 서비스 팩을 제거하는 등의 실행 애플리케이션에 수행되는 개별 변경을 롤백(rollback)할 수 있다. 블록(1016)에서, 실행 애플리케이션의 구성은 웹 프론트엔드(web frontends) 등에 배치된 애플리케이션의 부분을 부가 또는 제거함으로써 변경될 수 있다. 애플리케이션은 또한 애플리케이션 컴포넌트들 중 하나 이상의 더 새로운 버전을 설치함으로써 업그레이드될 수 있다.

[0156] **2.2 추상 객체와 관계 정의**

[0157] 추상 객체 정의는 설계 시간에 애플리케이션 구성을 검사한 후 배치하기 위하여 필요한 빌딩 블록을 정의하고 런타임에서 애플리케이션을 관리한다. 이들 빌딩 블록은 모델링된 시스템 내에 존재하는 엔티티를 나타낸다. 예를 들어, 우리는 추상 객체 정의를 사용하여 파일 및 디렉토리, 웹 서버 내의 구성 또는 sql 서버 내의 데이터베이스를 모델링한다.

[0158] 우리는 추상 관계 정의를 사용하여 추상 객체 정의 사이에서 발생할 수 있는 상호작용을 모델링한다. 관계는 바이너리이고 관계의 표명에 참여하는 인스턴스를 정의하는 객체 정의를 식별하도록 지시된다. 관계는 객체들을 함께 결합하는 방법을 제공하여 포함, 구성 및 객체간의 통신 링크를 모델링할 수 있다.

[0159] 제한은 그후 객체에 의해 사용되어 그들이 관계에 의해 참여하여 링크될 수 있는 객체를 제한하는 관계를 제한한다. 이들 제한은 정의 및 관계에 참여하는 세팅을 목표로 할 수 있다. 이것은 제한이 특정 정의로부터 도출되는 인스턴스로 관계의 참여자를 좁히도록 하고 인스턴스가 특정 범위내에 있는 세팅 값을 갖는 것을 요구한다.

[0160] 우리는 객체 정의를 세가지 카테고리, 즉, 시스템, 엔드포인트 및 리소스로 분할한다.

[0161] 추상 시스템 정의는 애플리케이션의 자기 포함 독립 배치가능 부분(self-contained independently deployable parts)을 설명하는 데 사용된다. 이들 정의는 프로세스 및 머신 경계를 횡단할 수 있는 잘 정의된 통신 채널을 통해 상호작용하는 애플리케이션의 부분을 나타낸다.

[0162] 추상 엔드포인트 정의는 시스템이 노출할 수 있는 통신 엔트포인트를 설명하는 데 사용된다. 이들은 설계 시간에 시스템 접속성을 검증하고 런타임에서 접속을 가능하게 하기 위하여 시스템이 알아야 하는 통신의 모든 형태를 모델링하는 데 사용된다.

[0163] 추상 리소스 정의는 시스템 내에 포함된 거동을 설명한다. 리소스 정의는 다른 리소스 정의에 대한 강한 종속성을 가질 수 있다. 이들 종속성은 특정 설치 순서를 요구하고 증명되지 않은 통신 메카니즘을 통해 런타임 상호작용을 개시하는 것을 포함할 수 있다.

[0164] 모든 추상 객체 정의는 세팅을 노출하는 능력을 공유한다. 이들 세팅은 세팅의 타입을 정의하기 위하여 xml 스키마를 사용하는 간단한 이름-값 쌍이다. 세팅은 동적이거나 정적일 수 있으며, 정적인 경우에는 배치 프로세스동안만 세팅될 수 있을 뿐이고, 동적인 경우에는 배치 후에 변경될 수 있다. 운영 시스템에 세팅 값을 적용하는 코드는 SDM 런타임 내에 호스팅된다.

[0165] SDM은 추상 객체 정의에 대하여 상속(inheritance)을 지원한다. 도출된 정의는 그 부모들에 의해 노출된 특성을 확장할 수 있고 그들 부모의 특성에 대한 값을 세팅할 수 있다. 도출된 정의는 참여자로서 그들의 부모를 식별하는 관계들 중의 임의의 것에 참여할 수 있다.

[0166] 관계 정의는 5가지의 카테고리, 즉, 통신, 포함, 델리게이션, 호스팅 및 참조로 분할된다.

[0167] 통신 관계는 추상 엔드포인트 정의 간의 잠재적인 통신 상호작용을 캡쳐하는 데 사용된다. 통신 관계의 존재는 시스템이 통신하기 위하여 식별된 정의의 엔드포인트를 노출하는 것이 가능함을 표시한다. 링크의 실제 확립은 엔드포인트에 대한 제한과 엔드포인트의 노출에 종속된다.

[0168] 포함 관계는 추상 객체 정의가 또다른 추상 객체 정의의 멤버를 포함하는 능력을 설명한다. 더 상세하게는, 2개의 추상 객체 정의 A와 B 간의 포함 관계는 A를 구현하는 구체적 객체 정의가 B를 구현하는 구체적 객체 정의의 멤버를 포함하도록 한다.

[0169] 우리는 포함을 사용하여 개발자가 애플리케이션을 작성할 때 발생하는 자연적 네스트된(nested) 구조를 모델링한다. 멤버 객체를 포함함으로써, 부모는 포함된 객체의 수명 및 가시성을 제어할 수 있다. 런타임 공간 내의

모든 객체 인스턴스는 다른 객체 인스턴스의 멤버로서 존재하고, 완전히 접속된 인스턴스의 세트를 형성한다. 따라서, 포함 관계의 세트는 인스턴스 공간 내에 발생하는 허용된 포함 패턴을 설명한다.

[0170] 델리게이션 관계는 포함된 객체 멤버를 선택적으로 노출하는 데 사용되고, 특히, 우리는 델리게이션을 사용하여 시스템 정의로부터 엔드포인트 멤버를 노출한다. 서브시스템으로부터 엔드포인트를 델리게이트함으로써, 외부 시스템은 프로토콜을 초월하는 구현을 노출하지 않고 특정 프로토콜에 의한 통신 능력을 노출한다.

[0171] 호스팅 및 참조 관계는 2가지 형태의 종속성 관계이다. 호스팅 관계는 구체적 객체의 인스턴스가 생성될 수 있기 전에 존재해야 하는 추상 객체 사이의 1차 종속성을 설명한다. 모든 인스턴스는 게스트로서 정확히 하나의 호스팅 관계에 참여해야 하고, 인스턴스 공간을 통해 완전히 접속된 트리를 형성하는 호스팅 관계를 발생시킨다. 참조 관계는 파라미터 플로우 및 구성 순서화를 위하여 사용될 수 있는 추가의 종속성을 캡쳐한다.

[0172] **2.3 구체적 객체 및 관계 정의**

[0173] 우리는 추상 객체 정의로부터 구체적 객체 정의를 작성하고 추상 관계 정의로부터 구체적 관계 정의를 작성한다.

[0174] 추상 객체 정의와 추상 관계 정의의 조합은 타겟 시스템을 모델링하기 위한 스키마를 정의한다. 구체적 객체 정의의 역할은 하나 이상의 추상 정의에 기초하여 재사용가능한 구성을 생성하도록 추상 정의 공간의 서브세트를 사용하는 것이다. 간단한 유추로서, 추상 정의 공간은 데이터베이스에 대한 스키마에 비교될 수 있고, 구체적인 객체 정의는 데이터베이스 내의 행의 세트에 대한 재사용가능 템플릿을 나타낸다. 행은 단지 구체적 객체의 인스턴스가 생성될 때 데이터베이스 내에 생성된다. 설계 시간 유효화를 수행하기 위하여, 우리는 스키마(예를 들어, 이질적인 키 등)의 제한에 대한 데이터베이스 내의 행을 유효화하는 것과 동일한 방법으로 추상 정의 공간에 대하여 구체적 객체 정의를 유효화할 수 있다.

[0175] 각각의 구체적인 객체 정의는 특정 추상 객체 정의에 구현을 제공한다. 구현은 세팅 스키마에 대한 확장, 세팅을 위한 값 및 객체 멤버를 위한 선언, 관계 멤버 및 제한 멤버 및 플로우 멤버를 포함한다. 구체적인 객체의 거동은 추상 객체의 정의를 따른다: 추상 시스템 정의는 구체적 시스템 정의가 되고, 추상 엔드포인트 정의는 구체적 엔드포인트 정의가 되고, 추상 리소스 정의는 구체적 리소스 정의가 된다.

[0176] 각각의 구체적 관계 정의는 특정 추상 관계 정의에 구현을 제공한다. 구현은 세팅 선언 및 값, 동일한 관계 카테고리의 네스트된 멤버(호스팅, 포함, 통신 등), 및 관계에 참여할 수 있는 타입에 관한 제한을 포함할 수 있다.

[0177] 구체적인 호스팅 관계는 하나의 구체적 객체의 멤버의 또다른 구체적 객체에 대한 맵핑을 정의하는 데 사용된다. 예를 들어, 구체적 호스팅 관계는 웹 애플리케이션과 배치될 IIS 호스트 간의 결합을 식별하는 데 사용될 수 있다. 하나 이상의 구체적 호스팅 관계는 개발자가 특정 토폴로지에 대한 다른 배치 구성을 정의하도록 하는 주어진 타입을 위하여 존재할 수 있다.

[0178] **2.4 멤버**

[0179] 구체적 타입은 다른 구체적 또는 추상 객체의 멤버를 선언할 수 있다. 우리는 이들을 객체 멤버라 부른다. 이들 멤버는 객체 멤버 사이의 관계를 정의하는 관계 멤버로부터 참조된다.

[0180] 객체 멤버는 특정 객체 정의의 인스턴스를 생성하는 데 사용된다. 세팅 플로우는 객체에 대한 값을 제공하는 데 사용된다. 객체 멤버를 선언할 때, 사용자는 객체 멤버가 외부 시스템이 생성되는 것과 동시에 생성되는지(값 시멘틱), 또는 나중에 발생하는 명시적인 새로운 동작에 의해 생성되는지(참조 시멘틱)를 결정할 수 있다.

[0181] 관계 멤버는 객체 멤버가 생성될 때 참여하는 관계를 정의한다. 객체 멤버가 그 부모에 의해 포함되면, 포함 관계 멤버는 타입 멤버와 외부 타입 사이에서 선언될 것이다. 객체 멤버가 델리게이트되면, 델리게이션 관계 멤버는 객체 멤버 및 네스트된 객체 멤버 사이에서 정의된다. 통신 관계 멤버는 객체 멤버 상의 엔드포인트 사이에서 선언될 수 있고 종속성 관계 멤버(참조 및 호스팅)는 객체 멤버 또는 네스트된 객체 멤버 사이에서 선언될 수 있다.

[0182] 관계 제한은 특정 객체가 자발적으로 참여하는 관계의 세트를 좁히는 데 사용된다. 이들은 특정 관계에 대한 제한 및 관계의 다른 엔드에서의 참여자에 대한 제한을 식별한다.

[0183] **2.5 인스턴스 공간**

[0184] SDM 런타임에 저장된 인스턴스 공간은 모델링된 시스템의 현재 상태를 반영한다. 런타임은 생성된 인스턴스의 완전한 레코드 및 이들 인스턴스들간의 관계를 포함한다. 각각의 인스턴스는 각각의 버전이 변경 요구에 링크된 관련된 버전 히스토리를 갖는다.

[0185] 새로운 인스턴스를 생성하는 프로세스는 변경 요구에 의해 개시된다. 변경 요구는 타입에 대한 생성, 갱신 및 삭제 요구의 세트 및 기존의 인스턴스의 특정 멤버와 관련된 관계를 정의하고, 루트(root)는 특별한 경우이다.

[0186] 변경 요구는 런타임에 의해 확장되고 모든 제한에 대하여 검증된 후 구성된다. 확장 프로세스는 객체를 포함하는 구성 요구의 부분으로서 암시적으로 구성되는 객체 및 관계 인스턴스를 식별하고 그후 세팅 플로우가 모든 관계에 걸쳐 평가된다. 검증 단계는 모든 요구된 관계들이 존재하는지, 그리고 그 관계들이 모든 제한을 이행하는지를 검사한다. 마지막으로, 구성 프로세스는 각각의 인스턴스의 배치, 갱신 또는 제어에 대하여 적절한 순서를 결정하고 그후 정확한 시퀀스로 각각의 인스턴스를 인스턴스 매니저에 전달하여 적절한 액션을 수행한다.

[0187] **2.6 계층화(layering)**

[0188] SDM 모델의 목표는 애플리케이션의 개발자, 소프트웨어 인프라스트럭쳐의 설계자 및 데이터센터의 기획자 간의 관심의 분리를 허용하는 것이다. 이들 그룹의 각각은 특정 서비스에 초점을 맞추고 다른 종속성 세트를 갖는다.

[0189] 예를 들어, 개발자는 주로 구성 및 SQP, IIS 및 CLR 등에 의존하는 호스트들간의 접속성에 대하여 관심을 갖는다. 호스트 구성의 설계자는 네트워크 토폴로지 및 OS 구성에 대하여 관심을 갖는 한편, 네트워크 토폴로지, OS 구성 및 저장 맵핑을 개발하는 기획자는 데이터센터에 존재하는 하드웨어에 대하여 알 필요가 있다.

[0190] 관심의 분리를 지원하기 위하여, SDM은 계층화의 개념을 노출한다. 계층화는 호스팅 관계를 사용하여, 애플리케이션이 종속하는 서비스들을 애플리케이션의 포함 구조의 부분으로서 선언하지 않고서, 애플리케이션을 그러한 서비스들에 결합시킨다.

[0191] 우리는 SDM 모델의 부분으로서 4개의 층을 식별한다.

[0192] 애플리케이션 층

[0193] • 애플리케이션 층은 제한된 컨텍스트 내에서 애플리케이션의 구성을 지원한다. 컨텍스트는 호스트 층 내에 식별된 호스트의 구성에 의해 정의된다.

[0194] • 애플리케이션 층 내의 시스템 정의의 예는 웹 서비스, 데이터베이스 및 비즈토크 스케쥴(biztalk schedules)을 포함한다.

[0195] 호스트 층

[0196] • 소프트웨어 컴포넌트들 중에서 데이터센터를 작성. 컴포넌트들 사이의 접속을 구성. 이들 컴포넌트들의 일부가 애플리케이션 층에 대한 호스트로서 활동한다.

[0197] • 이러한 층 내의 시스템 정의의 예 - IIS, SQL, AD, EXCHANGE, DNS 및 비즈토크(Biztalk).

[0198] 네트워크/OS/저장 층

[0199] • 데이터 센터 네트워크 및 플랫폼 작성. 네트워크 보안 모델 및 오퍼레이팅 시스템 플랫폼 구성을 구성. 저장 장치를 오퍼레이팅 시스템 구성에 부가.

[0200] • 이러한 층 내의 시스템 정의의 예 - VLAN, Windows, 필터, 저장장치

[0201] 하드웨어 층

[0202] 하드웨어 층은 데이터센터 내에 존재하는 머신의 타입 및 이들 머신들 사이에 존재하는 물리적 접속을 식별한다.

[0203] 도 11은 층 4 웹 애플리케이션을 층 3 웹 서버에 맵핑한 예를 나타낸다. 각층의 외부 박스는 시스템을 나타내고, 경계 상의 박스는 엔드포인트를 나타내고 내부의 박스는 리소스를 나타낸다. 우리는 호스팅 관계를 통해

이들 요소의 각각을 이하의 층의 호스트에 맵핑한다.

[0204] 시스템의 요구된 관계를 만족하기 위하여, 우리는 그 시스템을 일치 능력을 갖는 호스트 시스템에 결합한다. 우리는 이 프로세스를 플레이스먼트(placement)라 부른다. 설계시에, 우리는 가능한 플레이스먼트를 나타내는 구체적 호스트 관계를 구성한다. 배치시에, 우리는 구체적 호스팅 관계의 인스턴스를 실증하여 게스트 시스템 인스턴스를 호스트 시스템 인스턴스에 결합한다.

[0205] **2.7 모델 평가**

[0206] SDM 모델은 분산된 시스템으로의 변경을 관리하는 잘 정의된 프로세스와 관련된다.

[0207] 각각의 변경은 요구의 액션이 분산된 후 타겟 시스템에 대하여 실행되기 전에 몇개의 처리 단계를 통과한 선언적 변경 요구에 의해 유도된다.

[0208] **3. 구현 세부사항**

[0209] **3.1 명명**

[0210] SDM 내에는, 객체를 식별하는 강력한 명명 시스템을 필요로 하는 다수의 장소가 있다. 다음의 명명 시스템은, 정의의 사용자가, 해당 정의가 개발자가 원래 공표한 것과 동일하다는 것을 확인할 수 있는 방법으로 타입의 생성자가 정의에 서명할 수 있게 한다.

[0211] 다음의 헤더는 sdm 이름공간(namespace)에 대한 식별자의 예이다.

```
<sdm name="FileSystem"
      version="0.1.0.0"
      publicKeyToken="AAAABBBBCCCCDDDD"
      culture="neutral"
      platform="neutral"
      publicKey="aLongKey"
      signature="TheHashOfTheFileContents" >
  </sdm>
```

[0212]

[0213] 또다른 이름공간 내의 타입을 참조하기 위하여 이름공간을 임포트시킬 필요가 있다.

[0214]
```
<import alias="FileSystem" name="FileSystem" version="0.1.0.0" publicKeyToken="AAAABBBBCCCCDDDD"/>
```

[0215] 그후, 이름공간 내에서 타입을 지칭하는 별칭(alias)을 사용할 수 있다.

[0216]
```
FileSystem:file
```

[0217] **3.1.1 식별정보(Identity)**

[0218] SDM 이름은 그들이 정의되는 이름공간에 의해 제한된다. 이름공간은 이름, 버전, 언어 및 공중 키 토큰에 의해 식별되고 단일 파일 내에 포함된다.

[0219] 식별의 기본 형태는 이름, 버전, 컬쳐(culture), 플랫폼 및 공중 키 토큰을 포함한다.

```
<xs:attributeGroup name="Identity">
    <xs:attribute name="name" type="simpleName" use="required"/>
    <xs:attribute name="version" type="fourPartVersionType" use="required"/>
    <xs:attribute name="publicKeyToken" type="publicKeyTokenType" use="optional"/>
    <xs:attribute name="culture" type="xs:string" use="optional"/>
    <xs:attribute name="platform" type="xs:string" use="optional"/>
</xs:attributeGroup>
```

[0220]

[0221]

속성/요소	설명
name	sdm 파일의 이름은 개발자가 파일의 콘텐츠를 참조하는 데 사용할 수 있는 친근한 이름이다. 공중 키 토큰과 결합된 이름은 파일에 대한 강력한 이름을 제공한다.
version	버전은 파일의 콘텐츠의 버전을 식별하는 데 사용된다. 파일의 모든 요소는 동일한 버전 번호를 채택한다.
PublicKeyToken	공중 키 토큰은 파일과 관련된 공중 키에 대한 짧은 이름이다.
culture	바이너리의 컬쳐. 중립에 대한 디폴트.
platform	바이너리에 대한 지원된 플랫폼.

[0222] 기본 식별정보는 기존의 식별정보 또는 서명 및 공중 키를 결합하여 참조하는 데 사용되어, 새로운 강력한 식별정보를 생성한다. 문서는 개인 키를 사용하여 서명되고 문서의 사용자가 공중 키를 사용하여 그 콘텐츠를 검증하도록 허용한다.

```
<xs:attributeGroup name="namespaceIdentity">
    <xs:attributeGroup ref="Identity"/>
    <xs:attribute name="signature" type="xs:string" use="optional"/>
    <xs:attribute name="publicKey" type="xs:string" use="optional"/>
</xs:attributeGroup>
```

[0223]

[0224]

속성/요소	설명
signature	포함된 sdm 타입 정의의 서명된 해쉬(hash)
publickey	파일의 서명을 검사하는 데 사용될 수 있는 공중 키

[0225] 공중 키 토큰은 공중/개인 키 쌍의 공중 부분을 식별하는 16 문자 헥스 스트링(16 character hex string)이다. 이것은 공중 키가 아니고, 단순히 공중 키의 64 비트 해쉬이다.

```
<xs:simpleType name="publicKeyTokenType">
    <xs:annotation>
        <xs:documentation>Public Key Token: 16 hex digits in size</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-f]|[A-F]){16}"/>
    </xs:restriction>
</xs:simpleType>
```

[0226]

[0227] ### 3.1.2 버전

[0228] 파일 버전은 형태 N.N.N.N의 4개의 부분 번호에 의해 정의된다. 여기서, 0<=N<65535이다. 협약(convention)에 의해 번호는 Major.Minor.Build.Revision를 지칭한다.

```
<xs:simpleType name="fourPartVersionType">
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])(\.([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-
2][0-9]|6553[0-5])){3}"/>
    </xs:restriction>
</xs:simpleType>
```

[0229]

[0230] ### 3.1.3 간단한 이름

[0231] 간단한 이름은 영숫자 문자 및 제한된 구두점으로 구성된다. 이름은 수가 아닌 문자로 시작해야 한다.

```
<xs:simpleType name="simpleName">
    <xs:annotation>
        <xs:documentation>name of a type or member</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*"/>
    </xs:restriction>
</xs:simpleType>
```

[0232]

[0233] 우리는 식별자에 대한 C# 정의에 맞추는 계획을 세웠다; 적절한 섹션(2.4.2)이 이하에서 삽입되었다. 스펙은 다음에서 찾을 수 있다.

[0234] http://devidiv/SpecTool/Documents/Whidbey/VCSharp/Formal%20Language%20Sepcification/Csharp%20Language%Specification.doc

[0235] 우리는 sdm 모델 내에서 "@" 접두사를 갖는 이름을 지원하지 않음을 주의한다.

[0236] 이 섹션에 주어진 식별자의 규칙은, 언더스코어(underscore)가 초기 문자로서 허용되는 것(C 프로그래밍 언어에서 일반적임), 유니코드 이스케이프 시퀀스가 식별자 내에서 허용되는 것, "@" 문자가 접두사로서 허용되어 키워드가 식별자로서 사용되도록 하는 것을 제외하고, 정확하게 유니코드 표준 부록 15(Unicode Standard Annex

15)에 에 의해 추천된 것에 대응한다.

identifier:
available-identifier
@ ***identifier-or-keyword***

available-identifier:
An ***identifier-or-keyword*** **that is not a** ***keyword***

identifier-or-keyword:
identifier-start-character identifier-part-characters $_{opt}$

identifier-start-character:
letter-character
_ (the underscore character U+005F)

identifier-part-characters:
identifier-part-character
identifier-part-characters identifier-part-character

identifier-part-character:
letter-character
decimal-digit-character
connecting-character
combining-character
formatting-character

letter-character:
A Unicode character of classes Lu, Ll, Lt, Lm, Lo, or Nl
A ***unicode-escape-sequence*** **representing a character of classes Lu, Ll, Lt, Lm, Lo, or Nl**

combining-character:
A Unicode character of classes Mn or Mc
A ***unicode-escape-sequence*** **representing a character of classes Mn or Mc**

decimal-digit-character:
A Unicode character of the class Nd
A ***unicode-escape-sequence*** **representing a character of the class Nd**

connecting-character:
A Unicode character of the class Pc
A ***unicode-escape-sequence*** **representing a character of the class Pc**

formatting-character:
A Unicode character of the class Cf
A ***unicode-escape-sequence*** **representing a character of the class Cf**

[0237]

[0238] 상술한 바와 같은 유니코드 문자 클래스에 관한 정보에 대하여, 유니코드 표준 부록, 버전 3.0, 섹션 4.5를 참조하기 바란다.

[0239] 유효 식별자의 예는 "idenfifier1", "_identifier2", 및 "@if"를 포함한다. 합치 프로그램(conforming program) 내의 식별자는 유니코드 표준 부록 15에 의해 정의된 바와 같이 유니코드 정규화 양식 C(Unicode Normalization Form C)에 의해 정의된 정규의 포맷 내에 있어야 한다. 정규화 양식 C에 있지 않은 식별자와 마주칠 때의 거동은 구현에 따라 정의되지만, 진단은 요구되지 않는다.

[0240] 접두사 "@"는 식별자로서 키워드의 사용을 가능하게 하고, 다른 프로그래밍 언어와 인터페이스할 때 유용하다. 문자@는 실질적으로 식별자의 부분이 아니며, 따라서, 식별자는 접두사없이 정상 식별자로서 다른 언어로 표시될 수 있다. @ 접두사를 갖는 식별자는 축어적 식별자(verabtim idenfifier)라 불리운다. 키워드가 아닌 식별자를 위한 @ 접두사의 사용은 허용되지만, 스타일의 문제로서 강력하게 방해된다.

[0241] 예:

```
        class @class
{
        public static void @static(bool @bool) {
                if (@bool)
                        System.Console.WriteLine("true");
                else
                        System.Console.WriteLine("false");
        }
}
        class Class1
{
        static void M() {
                cl\u0061ss.st\u0061tic(true);
        }
}
```

[0242]

[0243] 는 "bool"으로 명명된 파라미터를 취하는 "static"으로 명명된 정적 방법(static method)을 갖는 "class"로 명명된 클래스를 정의한다. 유니코드 이스케이프(Unicode escape)는 키워드 내에서 허용되지 않으므로, 토큰"cl

\u0061ss"은 식별자이고, "@class"와 동일한 식별자이다.

[0244] 2개의 식별자가 다음의 변환이 순서대로 적용된 후에 동일한 경우, 이러한 2개의 식별자는 동일한 것으로 간주된다.

[0245] ◆ 접두사"@"는 사용되면 제거된다.

[0246] ◆ 각각의 unicode-escape-sequence는 그 대응하는 유니코드 문자로 변환된다.

[0247] ◆ 임의의 formatting-character들이 제거된다.

[0248] 2개의 연속된 유니코드 문자(U+005F)를 포함하는 식별자는 구현에 의해 사용하도록 저장된다. 예를 들어, 구현은 2개의 언더스코어로 시작하는 확장된 키워드를 제공한다.

[0249] **3.1.4 저장된 이름(Reserved names)**

[0250] 다음은 저장된 이름의 리스트이며, SDM 모델 내의 객체에 대한 이름을 생성할 때 사용자는 이러한 저장된 이름들을 이용할 수 없다.

[0251] 소정의 컨텍스트 내에서, 소정의 이름이 저장될 것이다.

[0252]

콘텍스트	이름
추상 및 구체적 정의	이것
추상 및 구체적 호스팅 관계 정의	게스트, 호스트
추상 및 구체적 포함 관계 정의	부모, 멤버
추상 및 구체적 통신 관계 정의	클라이언트, 서버
추상 및 구체적 참조 관계 정의	소스, 종속
추상 및 구체적 델리게이션 관계 정의	프록시, 델리게이트

[0253] 이들 이름은 CLR과의 통합 때문에 저장된다.

C# keywords				
AddHandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar	CDate
CDec	CDbl	Char	CInt	Class
CLng	CObj	Const	CShort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	ElseIf	End	Enum
Erase	Error	Event	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Function	Get	GetType	Goto
Handles	If	Implements	Imports	In
Inherits	Integer	Interface	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads	Overridable
Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw
To	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	eval	extends
instanceof	package	var		

[0254]

[0255] ### 3.1.5 다른 이름공간에 대한 참조

[0256] 우리는 이름공간을 현재의 이름공간에 임포트하고, 별칭을 이름공간에 관련시킴으로써, 이름공간이 다른 이름공간을 참조하는 것을 허용한다. 임포트된 이름공간은 이름, 버전 및 공중 키 토큰에 의해 참조된다. 버저닝은 섹션 3.16에 설명될 것이다.

```
<xs:complexType name="import">
    <xs:attribute name="alias" type="simpleName" use="required"/>
    <xs:attributeGroup ref="identity"/>
</xs:complexType>
```

[0257]

[0258]

속성/요소	설명
alias	별칭은 현재의 sdm 파일의 범위 내의 외부 sdm 파일을 참조하는 데 사용된다.

[0259] ### 3.1.6 인가된 경로(Qualified Path)

[0260] 인가된 경로는 현재의 이름공간 또는 별칭을 갖는 이름공간(aliased namespace)에 정의된 정의 또는 매니저로 지칭되는 이름이다.

[0261] [<alias>:]<simpleName>(.<simpleName>)*

[0262] 별칭은 임포트 명령문 내에 정의된다. 다음의 간단한 이름은 타입, 또는 경로의 경우에서의 네스트된 타입을 식별한다.

```
    <xs:simpleType name="qualifiedName">
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*(:[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*)?(\.[a-z,A-Z]{1}([0-9,a-z,A-
Z,_])*)*"/>
        </xs:restriction>
    </xs:simpleType>
```

[0263]

[0264] 3.1.7 정의 및 멤버 경로

[0265] 경로는 멤버 또는 세팅을 식별하는 이름의 시퀀스이다. 경로는 자신과 관련된 관계 또는 객체에 의해 정의된 공지된 이름 또는 멤버 이름으로 시작해야 한다.

[0266] <simpleName>(.<simpleName>)*

[0267]
```
<xs:simpleType name="path">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*(\.[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*)*"/>
    </xs:restriction>
</xs:simpleType>
```

[0268] 3.1.8 인스턴스 경로

[0269] 인스턴스 공간 내의 경로는 x경로에 기초하며, 이 때 x경로 내의 요소 이름은 멤버 이름에 대응하고, x경로 내의 속성은 세팅에 대응한다.

[0270] 3.1.9 이름 분석(Name Resolution)

[0271] 별칭으로 시작하지 않는 이름은 완전히 인가되지 않는다. 이것은 그들이 평가되는 범위가 결과적인 결합을 변경할 수 있는 것을 의미한다. 이 예는 네스트된 정의이다. 네스트된 정의 이름을 분석할 때, 로컬 범위 내의 정의는 더 넓은 범위 내의 정의를 은폐한다.

[0272] 3.2 세팅

[0273] 모든 정의는 세팅 선언을 노출시킬 수 있다. 이들 세팅은 구체적인 정의가 추상 정의로부터 생성될 때 또는 정의가 또다른 정의 내의 멤버로부터 참조될 때 제공될 수 있는 값을 설명하는 데 사용된다.

[0274] 세팅을 정의하기 위하여, 먼저 xsd를 사용하여 세팅의 정의를 정의하는 것이 필요하다.

[0275]
```
<xs:schema>
    <xs:simpleType name="registryValueType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="binary"/>
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="expandString"/>
            <xs:enumeration value="multiString"/>
            <xs:enumeration value="string"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

[0276] 그후, 그러한 정의를 사용하고 그 거동을 정의하기 위한 속성의 세트를 포함하는 세팅을 선언할 수 있다.

[0277]
```
<settingDeclaration name="valueType" type="registryValueType" access="readwrite" dynamic="false" required="true"/>
```

[0278] 일단 세팅 선언을 가지면, 세팅에 대한 값을 제공할 수 있다.

[0279]
```
<settingValue name="valueType" fixed="true">long</settingValue>
```

[0280] 3.2.1 세팅 정의

[0281] 우리는 XSD 스키마를 사용하여, 세팅을 선언함으로써 사용되는 세팅 정의를 정의한다. 비록 다른 스키마 요소가 심플 및 컴플렉스 타입의 사용을 지원하기 위해 존재하더라도, 우리는 하나의 스키마로부터 이러한 유형들의 사용을 지원할 수 있다.

[0282] 세팅 정의 섹션은 이름공간 선언 및 이름공간 임포트를 포함하는 완전한 xml 스키마를 포함해야 한다. 우리는 xsd 스키마 이름공간을 제외하고 xsd 스키마 내의 임포트가 sdm 파일 내의 임포트와 일치하는지를 검사할 것이다. 이것은 참조된 모든 타입이 또다른 sdm 파일 내에서 정의되어야 하는 것을 의미하고, 스키마는 보조적 xsd 파일 내에 정의되는 타입을 참조할 수 없다.

```
<xs:complexType name="settingDefinitions">
    <xs:sequence>
        <xs:element ref="xs:schema"/>
    </xs:sequence>
    <xs:attribute name="manager" type="qualifiedName" use="optional"/>
    <xs:attribute name="clrNamespace" type="xs:string" use="optional"/>
</xs:complexType>
```

[0283]

[0284]

속성/요소	설명
xs:schema	http://www.w3.org/2001/XMLSchma 이름공간 내의 스키마
manager	이 스키마를 위해 구조자 클래스(helper class)를 포함하는 clr 어셈블리
clrNameSpace	이들 클래스가 정의되는 clr 이름공간. 모든 세팅 타입은 CLR 직렬화를 통해 그들의 맵핑에 기초하여 clr로 분석될 것이다.

[0285] 세팅은 3개의 별개의 이름공간으로부터 분석가능해야 한다.

[0286] a) sdm 이름공간 - 우리가 시스템, 리소스, 엔드포인트, 관계, 제한 또는 플로우 타입 내의 세팅 타입을 지칭할 때.

[0287] b) clr 이름공간 - 우리가 clr 내의 강력한 타입 클래스를 사용하여 세팅을 지칭할 때 및 세팅 타입이 다른 세팅 타입 상에 작성될 때.

[0288] c) XSD 이름공간 - 세팅 타입이 다른 세팅 타입을 사용하여 작성될 때.

[0289] 이것이 제대로 행해지게 하기 위하여, 우리는 세팅을 선언하는 방법에 다수의 제한을 배치해야 한다.

[0290] a) 모든 세팅은 clr, sdm 및 xsd 이름공간의 각각의 동일한 그룹에 있어야 한다. 즉, 2개의 세팅이 함께 하나의 이름공간에 있으면, 그들은 3개의 모든 이름공간 내에 함께 있어야 한다.

[0291] b) xsd 스키마 정의 내의 임포트된 이름공간은 SDM 파일 내의 임포트된 이름공간 및 관련된 구조자 어셈블리 내의 임포트된 이름공간과 일치해야 한다.

[0292] c) xsd 이름공간을 제외하고, xsd 스키마 내의 모든 임포트된 이름공간은 sdm 파일 내에서 정의되어야 한다.

[0293] 임포트된 SDM 문서로부터의 XSD 타입은 QNames를 사용하여 액세스가능하다:

[0294]
```
<alias>:<type-name>
```

[0295] 따라서, 예를 들면, Foo.sdm이 Bar.sdm을 임포트시키면, Bar.sdm의 세팅 타입은 이 예가 다음과 같이 예시한 것처럼 Foo.sdm의 세팅 타입 요소 내에서 참조될 수 있다.

```
<!--Foo.sdm-->
...
<import alias="bar" location="Bar.sdm".../>

<settingTypes>
        ...
        <xs:simpleType name="D">
            <xs:restriction base="bar:B".../>
        </xs:simpleType>
        ...
</settingTypes>

<!--Bar.sdm-->
...
<settingTypes>
        ...
        <xs:simpleType name="B">
            <xs:restriction base="xs:string".../>
        </xs:simpleType>
        ...
</settingTypes>
```

[0296]

[0297] ### 3.2.2 내장 심플 데이터 타입(Built in simple data types)

[0298] SDM은 XSD와 C# 이름공간의 교차부에 있는 데이터 유형을 작성하는 제한된 세트를 지원한다. 이들 타입은 SDM 런타임에 의해 본래 지원되고 다음의 테이블 내에 정의된다. 이들 타입에 더하여, 사용자는 xsd 및 cls 타입

간의 맵핑을 자유롭게 구성하여 사용한다.

[0299]

타입	설명	XSD 타입	C# 타입
string	스트링은 유니코드 문자의 시퀀스이다.	string	string
integer	64 비트 부호의 정수 타입	long	long
float	단일 정밀도(single precision) 부동 소수점 타입	float	float
double	배정밀도(double precision) 부동 소수점 타입	double	double
boolean	불린 값은 참 또는 거짓이다.	boolean	boolean
any	모든 다른 타입의 기본 타입	any	object
Date	간단한 날짜	Date	dateTime

[0300] 이들 타입은 c# 및 xsd 타입 공간 내의 이들 타입의 호환가능한 도출(compatible derivations)로 플로우될 수 있다. 예를 들어, 스트링을 위한 값은 스트링 상의 한정을 정의한 xsd 타입으로 플로우되고, 임의의 값은 type="any"을 수락하는 세팅으로 플로우될 수 있다.

[0301] **3.2.2.1 XSD 내장 타입**

[0302] 도 12는 내장 데이터타입 계층의 일례를 나타낸다.

[0303] **3.2.2.2 C# 데이터 타입**

[0304]

타입	설명	예
object	모든 다른 타입의 궁극적인 기본 타입	object 0 = null;
string	스트링 타입; 스트링은 유니코드 문자의 시퀀스이다.	string s = "hello";
stype	8 비트 부호 표시 정수(signed integral) 타입	sbyte val = 12;
short	16 비트 부호 표시 정수 타입	short val = 12;
int	32 비트 부호 표시 정수 타입	int val = 12;
long	64 비트 부호 표시 정수 타입	long val1 = 12; long val2 = 34L;
byte	8 비트 부호 미표시 정수(unsigned integral) 타입	byte val1 = 12;
ushort	16 비트 부호 미표시 정수 타입	ushort val1 = 12;
uint	32 비트 부호 미표시 정수 타입	uint val1 = 12; uint val2 = 34U;

ulong	64 비트 부호 미표시 정수 타입	ulong val1 = 12;
		ulong val2 = 34U;
		ulong val3 =56L;
		ulong val4 = 78UL;

float	단일 정밀도 부동 소수점 타입	float val = 1.23F;
double	배정밀도 부동 소수점 타입	double val1 = 1.23; double val2 = 4.56D;
bool	불린 타입; bool 값은 참 또는 거짓임	bool val1 = true; bool val2 = false;
char	문자 타입; char 값은 유니코드 문자	char val = 'h';
decimal	28 상위 디지트를 갖는 정밀 십진 타입	decimal val = 1.23M;

[0305] **3.2.2.3 지원된 변환**

[0306] 이들은 xsd 타입 및 cls 타입간에 존재하는 변환이다.

[0307] XML 스키마 (XSD) 타입 .NET 프레임워크 타입

[0308]	anyURI	System.Uri
[0309]	base64Binary	System.Byte□
[0310]	Boolean	System.Boolean
[0311]	Byte	System.SByte
[0312]	Date	System.DateTime
[0313]	dateTime	System.DateTime
[0314]	decimal	System.Decimal
[0315]	Double	System.Double
[0316]	duration	System.TimeSpan
[0317]	ENTITIES	System.String□
[0318]	ENTITY	System.String
[0319]	Float	System.Single
[0320]	gDay	System.DateTime
[0321]	gMonthDay	System.DateTime
[0322]	gYear	System.DateTime
[0323]	gYearMonth	System.DateTime
[0324]	hexBinary	System.Byte□
[0325]	ID	System.String
[0326]	IDREF	System.String
[0327]	IDREFS	System.String□
[0328]	int	System.Int32
[0329]	integer	System.Decimal
[0330]	language	System.String
[0331]	long	System.Int64
[0332]	month	System.DateTime
[0333]	Name	System.String
[0334]	NCName	System.String
[0335]	negativeInteger	System.Decimal
[0336]	NMTOKEN	System.String
[0337]	NMTOKENS	System.String□
[0338]	nonNegativeInteger	System.Decimal
[0339]	nonPositiveInteger	System.Decimal
[0340]	normalizedString	System.String
[0341]	NOTATION	System.String
[0342]	positiveInteger	System.Decimal
[0343]	QName	System.Xml.XmlQualifiedName

[0344] short System.Int16

[0345] string System.String

[0346] time System.DateTime

[0347] timePeriod System.DateTime

[0348] token System.String

[0349] unsignedByte System.Byte

[0350] unsignedInt System.UInt32

[0351] unsignedLong System.UInt64

[0352] unsignedShort System.UInt16

[0353] ### 3.2.3 세팅 선언(Setting Declaration)

[0354] 세팅 선언 섹션은 이전의 섹션으로부터의 세팅 정의를 사용하여 명명된 세팅을 생성한다. 속성은 각각의 세팅에 관한 또다른 정보를 제공하기 위하여 사용된다.

[0355]
```
<xs:complexType name="SettingDeclaration">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="List" type="xs:boolean"/>
            <xs:attributeGroup ref="settingsAttributes"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0356]

속성/요소	설명
settingsAttributes	개별 세팅 선언에 적용될 수 있는 속성의 세트
list	이 속성은 이 세팅이 단일 값이 아닌 값의 리스트인 것을 표시하는데 사용된다.

[0357] ### 3.2.4 리스트 지원

[0358] 다치 세팅(multivalued settings)의 조작을 지원하기 위하여, 우리는 세팅 값의 간단한 리스트를 지원한다. 리스트는 세팅 선언과 동일한 정의의 값의 시퀀스이다. 리스트들은 그들이 대체되거나 병합될 수 있는 다른 리스트들로 플로우될 수 있다. 값을 리스트로 병합할 때 중복 검출(duplication detection)을 지원하지 않는데, 그 이유는 이것은 세팅 플로우를 사용하여 더 유연하게 수행될 수 있기 때문이며, 또한 순서화의 임의의 형태를 보증하지 않는다.

[0359] 리스트 선언은 참(true)으로 세팅된 속성 리스트를 포함한다.

[0360]
```
<settingDeclaration name="roles" type="xs:string" list="true"/>
```

[0361] 다음으로, 값은 settingValueList를 사용하여 제공된다. 리스트를 제공할 때, 사용자는 이전의 값으로 대체되거나 병합될지를 지정할 수 있다.

[0362]
```
<settingValueList name="roles" fix="true" replace="true">
    <value>staticContent</value>
    <value>aspPages</value>
</settingValueList>
```

[0363] sdm은 값의 리스트의 간단한 조작을 지원한다. 플로우 멤버로부터의 경로가 세팅 선언을 목적으로 할 때, 그 결과적인 거동은 경로의 어느 한쪽 끝에서의 정의에 종속된다.

[0364]

소스	목적지	결과
요소	리스트	대체=거짓 - 요소는 리스트에 부가된다. 대체=거짓 -단일 요소를 갖는 리스트

리스트	리스트	대체=거짓 - 소스 및 목적지 리스트가 병합된다. 대체=참 -소스 리스트
리스트	요소	sdm은 리스트로부터 어떤 요소가 선택되는지를 분석할 수 없으므로, 이 조합은 지원되지 않는다.

[0365] ### 3.2.5 세팅 속성

[0366] 세팅 속성은 런타임에 의해 사용되어 특정 세팅의 거동을 설명한다.

```
<xs:attributeGroup name="settingsAttributes">
    <xs:attribute name="access">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="readwrite"/>
                <xs:enumeration value="readonly"/>
                <xs:enumeration value="writeonly"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="secure" type="xs:boolean" />
    <xs:attribute name="deploymentTime" type="xs:boolean"/>
    <xs:attribute name="required" type="xs:boolean"/>
    <xs:attribute name="dynamic" type="xs:boolean"/>
    <xs:attribute name="keyValue" type="xs:boolean"/>
    <xs:attribute name="nillable" type="xs:boolean" />
</xs:attributeGroup>
```

[0367]

[0368]

속성 이름	설명	디폴트
access	세팅의 액세스 속성은 세팅의 값을 판독 및 기록하는 것이 허용되는지를 지정한다; SDM 런타임 액세스 제어를 제공하고 설계자에게 규칙을 표시/편집한다.	판독 기입

<table>
<tr><td></td><td>
<table>
<tr><td>속성 값</td><td>SDM내의 의미 런타임</td><td>표시/편집 규칙</td></tr>
<tr><td>판독기입</td><td>세팅값이 판독되고 기입될 수 있는 것을 표시</td><td>값이 표시되고 편집될 수 있다.</td></tr>
<tr><td>판독전용</td><td>세팅값이 판독될수 있지만, 기입될 수 기입될 수 없는 것을 표시. 이 값은 플로우를 위한 타겟일 수 없다.

일반적으로, 판독전용 세팅은 실세계 인스턴스에 의해 제공되거나 계산될 수 있는 것이다.

예: 서버 상의 접속의 수를 나타내는 값.</td><td>값이 표시되지만, 편집될 수 없다.</td></tr>
<tr><td>기록전용</td><td>세팅 값이 기입되지만, 판독되지 않음을 표시. 이 값은 플로우를 위한 소스일 수 없다.

예: 서비스 어카운트를 위한 패스워드</td><td>값이 마스크되지만, 편집될 수 있다.</td></tr>
</table>
</td><td></td></tr>
<tr><td>deploymentTime</td><td>인스턴스를 위한 배치 프로세스의 부분으로서만 제공될 수 있는 값. 설계 시간 제한은 이 값에 대하여 평가될 수 없다. 정의 또는 멤버는 이 세팅에 대한 고정 값을 공급할 수 없다.</td><td>거짓</td></tr>
<tr><td>required</td><td>required가 참이면, 인스턴스가 배치되기 전에 값은 이 세팅에 제공되어야 한다. 이 세팅은 판독전용일 수 없다.</td><td>거짓</td></tr>
<tr><td>dynamic</td><td>dynamic이 참이면, 인스턴스가 배치된 후에, 인스턴스 매니저는 이 값에 대한 변경을 지원한다.</td><td>참</td></tr>
<tr><td>keyValue</td><td>Keyvalue는 그 호스팅 범위 내에서 세팅이 고유하다는 것을 표시하는 데 사용된다. 인스턴스 매니저는 경로 내의 이 세팅을 사용하여 객체 인스턴스를 식별하고 기존의 인스턴스와의 충돌을 검출한다</td><td>거짓</td></tr>
<tr><td>Secure</td><td>세팅의 secure 속성은 SDM 문서에 저장될 때 세팅의 값이 암호화(참 또는 거짓)되어야 하는지를 지정한다. 또한 설치 등의 조작 동안 툴이 이 값을 로그해야 하는지를 표시.</td><td>거짓</td></tr>
<tr><td>nillable</td><td>세팅의 nillable 속성은 이름공간http:www.w3.org/2002/XMLSchema-instance로부터 이름공간-인가된 속성xsi:nil을 전달하면, 세팅 값이 유효한지를 표시</td><td>거짓</td></tr>
</table>

[0369] **3.2.6 세팅 값**

[0370] 세팅이 단일 값으로서 선언되는지 또는 리스트로서 선언되는지에 따라, 세팅에 대한 값은 세팅 값 요소 또는 세팅 값 리스트 요소를 사용하여 제공될 수 있다.

[0371] **3.2.6.1 세팅 값**

[0372] 세팅 값은 특정 세팅 선언을 위한 값을 제공하는 데 사용된다. 값은 선언과 관련된 정의를 일치해야 한다. 값이 선언 고정되면, 제공된 값은 값이 고정된 포인트에 의존하여 모든 도출된 정의 또는 참조 멤버에 사용될 것

이다. 값이 고정되면 오버라이드(override)될 수 없다.

```
<xs:complexType name="settingValue">
    <xs:complexContent>
        <xs:restriction base="xs:anyType">
            <xs:attribute name="name" type="simpleName" use="required"/>
            <xs:attribute name="fixed" type="xs:boolean" use="optional" default="false"/>
            <xs:attribute ref="xsi:nil" use="optional" default="false"/>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

[0373]

[0374]

속성/요소	설명
name	이 값이 적용될 세팅 선언의 이름
fixed	고정(fixed) 속성은 제공된 값이 새로운 값에 의해 후속으로 오버라이드될 수 있는지를 제어한다. 참의 고정 값은 세팅에 제공된 값이 오버라이드될 수 없다는 것을 표시한다. 고정 값이 구체적인 정의 멤버의 세팅 값에 대하여 참이면, 그 멤버의 모든 배치에 고정된다. 그렇지 않고, 고정 값이 거짓이면, 그 멤버의 각각의 배치에 오버라이드 가능이다. 고정 값이 구체적인 정의의 세팅 값에 대하여 참이면, 그 구체적인 정의의 모든 멤버(즉, 사용)에 대하여 고정된다. 그렇지 않고, 고정 값이 거짓이면, 그 구체적인 정의의 각각의 멤버(즉, 사용)로 변경될 수 있다. 고정 값이 추상 정의의 세팅 값에 대하여 참이면, 그 추상 정의를 확장하는 추상 객체 또는 구현하는 구체적인 정의에 대하여 고정된다. 그렇지 않고, 고정 값이 거짓이면, 도출된 추상 정의 또는 멤버 선언에서 구체적인 정의에 의해 오버라이드될 수 있다.
Nil	마지막으로, 값 true를 갖는 속성xsl:nil 을 가지면, 세팅 값은 콘텐츠 없이 유효한 것으로 간주된다. 레이블된 요소가 비어야 하지만, 세팅 정의에 의해 허용되면 속성을 가질 수 있다.

[0375] 3.2.6.2 세팅 값 리스트

[0376] 세팅 값 리스트는 리스트로서 선언된 세팅을 위한 하나 이상의 값을 제공하는 데 사용된다. 값을 선언할 때, 사용자는 이전의 값과 병합하거나 이전의 모든 값을 중복 기입하도록 결정할 수 있다.

```
<xs:complexType name="settingValueList">
    <xs:sequence>
        <xs:element name="value" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:complexContent>
                    <xs:restriction base="xs:anyType">
                    </xs:restriction>
                </xs:complexContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="simpleName" use="required"/>
    <xs:attribute name="fixed" type="xs:boolean" use="optional" default="false"/>
    <xs:attribute name="replace" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
```

[0377]

[0378]

속성/요소	설명
name	이 값이 적용될 세팅 선언의 이름

fixed	고정(fixed) 속성은 제공된 값이 새로운 값에 의해 후속으로 오버라이드될 수 있는지를 제어한다. 참의 고정 값은 세팅에 제공된 값이 오버라이드될 수 없다는 것을 표시한다. 고정 값이 구체적인 정의 멤버의 세팅 값에 대하여 참이면, 그 멤버의 모든 배치에 고정된다. 그렇지 않고, 고정 값이 거짓이면, 그 멤버의 각각의 배치에 오버라이드 가능이다. 고정 값이 구체적인 정의의 세팅 값에 대하여 참이면, 그 구체적인 정의의 모든 멤버(즉, 사용)에 대하여 고정된다. 그렇지 않고, 고정 값이 거짓이면, 그 구체적인 정의의 각각의 멤버(즉, 사용)로 변경될 수 있다. 고정 값이 추상 정의의 세팅 값에 대하여 참이면, 그 추상 정의를 확장하는 추상 객체 또는 구현하는 구체적인 정의에 대하여 고정된다. 그렇지 않고, 고정 값이 거짓이면, 도출된 추상 정의 또는 멤버 선언에서 구체적인 정의에 의해 오버라이드될 수 있다.
replace	대체(replace) 속성은 세팅을 위한 새로운 값이 세팅을 위한 이전의 고정되지 않은 값으로 대체하거나 병합해야 하는지를 표시하는 데 사용된다. 대체가 참이면, 이전의 모든 값은 무시될 것이다. 대체가 거짓이면, 새로운 및 이전의 값은 단일 리스트로 결합될 것이다. 중복은 병합 프로세스 동안 검출되지 않을 것이다.

[0379] **3.2.7 세팅 상속(Setting Inheritance)**

[0380] 세팅 상속은 도출된 정의가 암시적으로 기본 정의로부터 세팅 선언 모두를 포함하는 것을 의미한다. 세팅 상속의 임의의 중요한 형태는 다음과 같다.

[0381] • 세팅 상속은 옮겨가는 것이다(transitive). C가 B로부터 도출되고 B가 A로부터 도출되면, C는 B 내에 선언된 세팅 뿐만 아니라 A 내에 선언된 세팅을 상속한다.

[0382] • 도출된 정의는 기본 정의로부터 상속되는 것에 새로운 세팅 선언을 부가할 수 있다. 이것은 확장되지만, 상속된 세팅의 정의를 제거할 수 없다.

[0383] **3.2.8 타입 변환**

[0384] 우리는 내장 타입들 사이의 손실없는 변환을 지원한다. 다른 타입 변환은 적절한 변환을 실행하기 위하여 플로우를 필요로 한다.

[0385] **3.3 속성**

[0386] SDM 내의 객체의 다수는 객체의 거동을 코어(core)하기 위하여 직교인 거동을 캡처하도록 귀착될 수 있다. 우리는 다음과 같이 정의된 일반적인 속성 모델을 사용한다.

[0387] **3.4 정의 및 멤버**

[0388] **3.4.1 정의**

[0389] 정의는 객체, 관계, 제한 및 플로우 정의가 도출되는 베이스이다. 모든 정의는 세팅 스키마 및 설계 표면 데이터를 포함할 수 있다. 각각의 정의는 간단한 이름에 의해 식별되고 매니저를 참조한다. 매니저는 이 특정 정의에 대한 SDM 런타임에 확장 지원을 제공한다.

[0390] 세팅 스키마는 이 정의의 인스턴스에서 찾을 수 있는 값을 정의한다. DesignData 요소는 설계 표면 상의 이 정의의 표시 및 편집에 특정인 데이터를 포함시키는 데 사용된다.

```
<xs:complexType name="Definition">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingDeclaration" type="SettingDeclaration"/>
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Manager" type="QualifiedName" use="optional"/>
    <xs:attribute name="ClrClassName" type="xs:string" use="optional"/>
</xs:complexType>
```

[0391]

[0392]

속성/요소	설명
SettingDeclaration	세팅의 선언
SettingValue	정의에 대한 세팅 또는 그 기본 정의에 대한 값. 값이 정의 내의 세팅 선언을 위해서 한번 제공될 수 있다.
SettingValueList	정의 또는 그 기본 정의 상에 기록가능한 리스트 세팅에 대한 값의 리스트
DesignData	설계 표면 특정 데이터
name	포함하는 sdm 파일의 범위내에서 고유한 이 정의에 대한 이름
Manager	이 정의에 대한 매니저 선언에 대한 참조. 매니저의 정의에 대한 섹션:3.10 참조
CirClassName	런타임 내의 이 정의를 지원하는 clr 클래스의 이름. 클래스는 매니저에 의해 식별되는 어셈블리 내에 존재해야 한다. 이 속성이 부여되면, 매니저 속성이 부여되어야 한다.
Description	정의의 텍스트 설명

[0393] ### 3.4.2 멤버

[0394] 멤버는 런타임에서 존재할 수 있는 정의 인스턴스를 식별하는 데 사용된다. 모든 멤버는 타입의 범위 내의 고유한 이름에 의해 식별되고, 멤버가 참조하는 정의에 대한 세팅을 제공할 수 있고, 설계 표면 특정 데이터를 포함할 수 있다.

```
<xs:complexType name="Member">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Definition" type="QualifiedName" use="required"/>
</xs:complexType>
```

[0395]

[0396]

속성/요소	설명
Description	멤버의 설명
DesignData	멤버에 대한 설계 표면 특정 정보
SettingValue	참조된 타입에 대한 기록가능 세팅에 대응하는 세팅의 값. 이들 값이 고정되도록 마크되면, 값은 인스턴스가 멤버를 위해 생성될 때 사용되어야 하며, 값이 고정되지 않으면,값은 배치 또는 플로우 파라미터에 의해 오버라이드될 수 있다.
SettingValueList	참조된 타입에 대한 기록 가능 세팅을 위한 값의 리스트
Name	포함 타입의 범위 내의 멤버에 대한 고유의 이름
Definition	이 멤버가 참조하는 정의의 이름

[0397] ## 3.5 세팅 플로우

[0398] 세팅 플로우는 객체 정의의 멤버들 사이에서, 그리고 관계 내의 참여자들 사이에서 파라미터를 전달하는 데 사용된다. 플로우의 부분으로서, 사용자는 변환(transformation)을 사용하여 세팅 값을 결합하거나 분리하고 새

로운 세팅 값을 산출한다.

[0399] 모든 세팅 플로우 멤버가 플로우 정의를 사용하여 변환을 구현한다. 플로우 정의는 sdm 파일 내에서 선언된다. 다음은 url을 구문 분석(parse)하는 플로우 타입이다.

```
<FlowDefinition name="UrlToComponents">
    <SettingDeclaration name="url" type="url" access="writeonly"/>
    <SettingDeclaration name="protocol" type="xs:string" access="readonly"/>
    <SettingDeclaration name="server" type="xs:string" access="readonly"/>
    <SettingDeclaration name="path" type="xs:string" access="readonly"/>
    <SettingDeclaration name="file" type="xs:string" access="readonly"/>
</FlowDefinition>
```

[0400]

[0401] 플로우 멤버는 그후 객체 또는 관계 내에서 선언된다. 플로우 멤버는 플로우 정의에 대한 입력을 제공하고, 그 후 플로우로부터 타겟 세팅으로 출력을 보낸다.

```
<Flow name="deconstructUrl"     type="UrlToComponents">
    <Input name="url" path="webservice.url"/>
    <Output name="server" path="webservice.server"/>
</Flow>
```

[0402]

[0403] ### 3.5.1 플로우 정의

[0404] 우리는 플로우 정의를 사용하여 우리가 세팅 값의 세트에 적용하고자 하는 특정 변환을 정의한다. 플로우 정의는 입력 세팅(기록 전용 세팅) 및 출력 세팅(판독 전용 세팅)을 정의하는 세팅 스키마, 변환을 정의하는 입력 인터페이스 등의 설계 표면 특정 정보에 대한 설계데이터 섹션 및 sdm 파일을 브라우징(browsing)할 때 사용하기 위한 설명을 노출시킨다. 플로우 정의는 그것이 정의되는 이름공간 내의 이름에 의해 식별된다. 정의는 또한 플로우를 평가할 때 런타임을 지원하는 매니저를 식별한다.

[0405] 우리는 직진 변환(straightforward transformations)이 요구되는 플로우 요소의 구성을 간략화시키기 위하여, 런타임이 몇개의 표준 플로우 정의를 포함할 것을 기대한다. 예는 카피, 병합 및 스트링 대입(substitution)을 포함할 수 있다. 플로우 정의가 파라미터화될 수 있으므로, 우리는 또한 구성 파라미터에 기초하여 다른 액션을 수행하는 간단한 하나 이상의 변환이 있음을 기대한다.

```
<xs:complexType name="FlowDefinition">
    <xs:complexContent>
        <xs:extension base="Definition"/>
    </xs:complexContent>
</xs:complexType>
```

[0406]

[0407] ### 3.5.2 플로우 멤버

[0408] 각각의 플로우 요소는 하나 이상의 소스 노드, 하나 이상의 목적지 노드, 임의의 정적 세팅 및 플로우 정의를 식별한다, 플로우가 평가될 때, 소스 데이터는 소스 노드로부터 수집되고 플로우 요소로부터의 세팅과 결합되고 변환을 위한 플로우 정의로 전달된다. 출력 데이터는 정의 노드로 전달된다.

[0409] 플로우의 재평가는 소스 값 중의 어느 하나가 변경될 때마다 트리거될 것이다. 이 때문에, 우리는 값을 플립플롭하는 순환 플로우를 회피할 필요가 있다. 값이 일정한 채로 유지되면, 루프는 종료할 것이다. 런타임은 스택 깊이의 추적을 유지하면서 무한 루프를 검출하고 종료할 것이다.

```
<xs:complexType name="FlowMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="SettingTarget"/>
                <xs:element name="Output" type="OutputPath"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0410]

[0411]

속성/요소	설명
input	플로우로의 입력으로서 사용되는 세팅 값으로의 경로의 리스트. 각각의 입력은 플로우 정의에 대한 기록 전용 세팅을 식별해야 한다.
output	이 플로우의 결과로서 세팅될 세팅으로의 경로의 리스트. 각각의 출력은 플로우 정의에 대한 판독 전용 세팅을 식별해야 한다.

[0412] ### 3.5.3 세팅 타겟

[0413] 세팅 타겟은 플로우가 정의된 컨텍스트 내의 잘 알려진 이름에 관련된 멤버 또는 네스트된 멤버 내의 세팅 값으로의 경로를 식별한다. 잘 알려진 이름의 예는정의 또는 기준 선언 내의 this, 호스팅 관계 선언 내의 host 및 guest, 또는 제한 선언 내에서 정의된 target를 포함한다. 세팅 타겟은 또한 경로에 의해 식별된 세팅의 목적지 세팅 또는 소스 값으로서 사용될 관련된 플로우 정의에 대한 세팅을 식별한다.

```
<xs:complexType name="SettingTarget">
    <xs:attribute name="Name" type="SimpleName"/>
    <xs:attribute name="Path" type="Path"/>
</xs:complexType>
```

[0414]

[0415]

속성/요소	설명
Name	경로에 의해 식별된 세팅의 소스 또는 목적지인 플로우 또는 제한 정의에 대한 세팅의 이름
Path	플로우가 정의된 컨텍스트에 관련된 소스 또는 목적지 세팅으로의 경로. 경로는 단일 세팅을 식별해야 한다. - 이것은 경로 상의 모든 멤버의 최대 카디널리티(cardinality)가 1 또는 0이어야 한다는 것을 의미한다.

[0416] 출력 경로는 타겟 값을 고정하고 대체하는 시멘틱을 지원하는 세팅 타겟에 대한 변수이다.

```
<xs:complexType name="OutputPath">
    <xs:complexContent>
        <xs:extension base="SettingTarget">
            <xs:attribute name="Fix" type="xs:boolean" use="optional"/>
            <xs:attribute name="Replace" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0417]

[0418]

속성/요소	설명
Fix	이것은 타겟 값이 고정되는 것을 선언할 것이다. 이것은 값을 후속으로 변경하려는 임의의 시도가 에러를 발생시키는 것을 의미한다.
Replace	리스트가 경로의 타겟일 때, 우리는 리스트의 현재 콘텐츠를 병합하거나 대체할 것을 선택할 수 있다. 디폴트 거동은 현재의 콘텐츠와 병합되는 것이다.

[0419] ### 3.6 세팅 제한(Settings Constraints)

[0420] 제한은 정의의 멤버의 세팅 값 또는 관계 내의 참여자에 대한 한정을 식별하는 데 사용된다. 이들 한정은 설계 시간 및 배치 시간에 인스턴스 공간 내에서 평가된다.

[0421] 모든 세팅 제한은 제한 정의를 사용하여 세팅 값을 평가한다. 제한 정의는 세팅 선언을 사용하여 그것이 제한하는 값을 식별한다. 다음의 제한 정의는 2개의 독립 변수(argument) 및 연산자를 취하는 간단한 비교 함수를 구현하고, 그후 제한을 평가하고, 마지막으로 성공 또는 에러를 리턴한다.

```
<ConstraintDefinition name="SimpleOperatorComparison">
    <SettingDeclaration name="LHS" type="xs:any" access="writeonly"/>
    <SettingDeclaration name="operator" type="operator" access="writeonly"/>
    <SettingDeclaration name="RHS" type="xs:any" access="writeonly"/>
</ConstraintDefinition>
```

[0422]

[0423] 제한 멤버는 평가에 대한 제한 타입에 값을 제공하는 데 사용된다.

```
<Constraint name="constraintSecurityMode" definition="SimpleOperatorComparison">
    <Input name="LHS" path="webservice.securityMode"/>
    <SettingValue name="operator">==</settingValue>
    <SettingValue name="RHS">basicAuth</settingValue>
</Constraint >
```

[0424]

[0425] ### 3.6.1 제한 정의

[0426] 제한 정의는 입력 값의 세트에 동작하는 제한을 정의한다. 제한은 파라미터화되어 고객 거동을 선택하거나 파라미터를 사용하여 그 거동을 정의하는 간단한 제한 엔진에 대하여 지원할 수 있다. 우리는 표준 제한 정의의 세트가 간단한 파라미터 값 제한 및 컴플렉스 제한의 세트에 대하여 기록되어 추상 객체 사이의 공지된 관계를 지원하는 것을 기대한다.

```
<xs:complexType name="ConstraintDefinition">
    <xs:complexContent>
        <xs:extension base="Definition"/>
    </xs:complexContent>
</xs:complexType>
```

[0427]

[0428] ### 3.6.2 제한 멤버

[0429] 제한 멤버는 특정 제한 정의에 대한 입력 값의 세트를 식별한다. 멤버는 세팅 값에 대한 정적 값을 식별할 수 있고, 입력 명령문을 사용하여 경로에 제한 세팅을 결합시킬 수 있다.

```
<xs:complexType name="ConstraintMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="SettingTarget"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0430]

[0431]

속성/요소	설명
input	제한으로의 입력 리스트. 입력은 제한에 전달될 소스 세팅 값과 결과로서 세팅될 제한 세팅으로의 경로를 식별한다. 소스 세팅 정의와 제한 세팅 저의는 호환가능해야 한다.

[0432] ### 3.7 시스템 엔드포인트 및 리소스 정의

[0433] 이 섹션은 추상 및 구체적 객체 정의에 대한 스키마를 설명한다.

[0434] 추상 객체 정의는 세팅 선언 세트를 나타내고, 자신이 참여하는 관계에 대한 제한을 포함할 수 있으며, 런타임 내에 관련 매니저를 갖는다.

[0435] 다음은 웹 서버에 대한 추상 시스템 정의이다. 웹 서버는 2개의 세팅을 가지며 적어도 vsite 타입에 포함하는 것을 요구하는 관계 제한을 갖는다.

```
<AbstractSystemDefinition name="WebServer"
                          clrClassName="micorosft.sdm.IISSupportClass"
                          manager="IISSupportCode">

    <SettingDeclaration name="serverName" type="xs:string"/>
    <SettingDeclaration name="category" type="xs:string"/>

    <RelationshipConstraint name="containsVsites"
                            relationship="containmentRelationship"
                            myRole="parent"
                            targetType="vsite"
                            minOccurs="1"
                            maxOccurs="unbounded" />

</ AbstractSystemDefinition >
```

[0436]

[0437] vsite는 서버 결합 정보를 포함하는 추상 엔드포인트 정의이다.

```
<AbstractEndpointDefinition name="vsite">
    <SettingDeclaration name="ipAddress" type="ipaddress" required="true"/>
    <SettingDeclaration name="Endpoint" type="xs:int"/>
    <SettingDeclaration name="domainName" type="xs:int"/>
    <SettingDeclaration name="securityModel" type="securityModelEnum"/>
</AbstractEndpointDefinition>
```

[0438]

[0439] 프론트엔드 웹서버에 대한 구체적 시스템 정의는 정적 콘텐츠로서 웹서버 카테고리를 식별하고, 1과 100 사이의 엔드포인트 인스턴스를 표현할 수 있는 단일 byReference 엔드포인트를 포함한다. 엔드포인트에 대한 구체적인 엔드포인트는 시스템 정의 내에 네스트되고, vsite에 대한 엔드포인트가 엔드포인트 80이 되는 것을 정의한다.

```
<SystemDefinition name="FrontendWebServer" implements="WebServer" >
    <SettingValue name="category" fixed="true">staticContentOnly</settingValue>
    <Port name="contentOnlyVsite" type="port80Vsite" isReference="true" minOccurs="1" maxOccurs="100"/>
    <PortDefinition name="port80Vsite" implements="vsite">
        <SettingValue name="Endpoint" fixed="true">80</settingValue>
    </PortDefinition >
</SystemDefinition>
```

[0440]

[0441] ### 3.7.1 객체 정의

[0442] 추상 및 구체적 객체는 다음의 베이스 객체 정의를 확장한다. 베이스 타입 정의의 요소에 더하여, 객체가 참여하는 관계를 제한하는 능력을 공유한다.

```
<xs:complexType name="ObjectDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Flow" type="FlowMember" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0443]

[0444]

속성/요소	설명
Flow	플로우 멤버 선언
RelationshipConstraint	이들 타입이 참여할 수 있는 관계에 대한 제한
RelationshipConstraintGroup	이들 타입이 참여할 수 있는 관계에 대한 제한

[0445] ### 3.7.2 추상 객체 정의

[0446] 추상 객체 정의는 설계 표면이 노출되고 모든 구체적 객체가 도출되는 빌딩 블록을 정의하는 데 사용된다: 구체적 객체 정의는 추상 객체 정의를 구현해야 한다.

[0447] 추상 객체 정의는 간단한 상속을 부가함으로써 SDM 객체를 확장한다: 확장 속성은 추상 객체 정의에 대한 베이스 객체 정의를 식별하는 데 사용된다. 추상 객체 정의는 그후 베이스 객체 정의로부터 세팅 및 관계 제한을 상속한다. 상속을 통해, 객체 정의는 새로운 세팅 및 제한을 부가함으로써 추상 객체 정의의 세팅 및 제한을 확장할 수 있다.

[0448] 추상 객체 정의는 또한 그들이 참여하기를 원하는 관계에 대한 제한을 부가할 수 있다. 예를 들어, 추상 객체 정의는 소정의 관계의 존재를 요구할 수 있거나, 관계의 다른 엔드 상에 배치될 수 있는 객체 정의를 제한할 수 있거나, 주어진 관계에 참여하는 인스턴스에 대한 세팅을 제한할 수 있다.

[0449] #### 3.7.2.1 추상 객체 정의

[0450] 모든 추상 객체는 그들이 관련되기를 원하는 층을 식별할 수 있다. 이것이 제공되지 않으면, 객체 정의는 임의의 층에 사용될 수 있는 것으로 가정한다. 추상 객체 정의는 그들의 확장하는 베이스 객체 정의를 식별할 수 있고, 어떤 경우, 그 객체 정의의 세팅과 제한을 상속하고, 베이스 객체 정의가 참여하는 관계 내의 베이스 객체 정의에 대하여 대체될 수 있다.

```
<xs:complexType name="AbstractObjectDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:attribute name="Layer" type="xs:string" use="optional"/>
            <xs:attribute name="Extends" type="QualifiedName" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0451]

[0452]

속성/요소	설명
layer	층(layer) 속성은 이 추상 객체 정의가 사용될 수 있는 층을 식별한다. 이것이 제공되지 않으면, 추상 객체 정의가 임의의 층에서 사용될 수 있다.
extends	이 객체 정의가 도출되는 추상 객체 정의를 식별

[0453] 3.7.2.2 추상 엔드포인트, 시스템 및 리소스 객체 정의

[0454] SDM 모델 내에 추상 객체 정의의 3가지 분류 - 즉, 추상 엔드포인트 정의, 추상 시스템 정의 및 추상 리소스 정의가 있다. 이들의 각각은 추상 객체 정의의 간단한 재명명이다.

```
<xs:complexType name="AbstractEndpointDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="AbstractSystemDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="AbstractResourceDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>
```

[0455]

[0456] 엔드포인트 정의는 통신 엔드포인트를 나타낸다. 엔드포인트에 대한 세팅은 결합 프로세스 내에서의 그 사용에 관련된다. 예를 들어, 클라이언트 서버 프로토콜로, 서버 엔드포인트 정의는 세팅 스키마를 사용하여 엔드포인트에 결합될 필요가 있는 세팅을 식별하고, 클라이언트 엔드포인트 정의는 클라이언트 특정 접속 속성을 노출한다.

[0457] 시스템 정의는 데이터, 소프트웨어 또는 하드웨어 요소의 수집을 나타내는 데 사용된다. 예는 웹 서비스, 데이터베이스 및 스위치를 포함한다. 리소스 정의는 시스템 정의의 부분으로서 식별될 수 있는 특정 요소를 캡쳐하는 데 사용된다.

[0458] 3.7.3 암시적 베이스 정의(Implicit base definitions)

[0459] 또다른 추상 객체 정의를 확장하지 않는 모든 추상 객체 정의는, 도 13에 도시된 바와 같이 엔드포인트, 시스템 또는 리소스 베이스 정의 중의 하나를 암시적으로 확장한다. 이들 베이스 정의는 관계 및 제한 선언에 사용될 수 있는 트리의 각각에 대한 루트를 형성한다. 이것은 관계 또는 제한으로 하여금 루트로부터 도출된 임의의 타입이 식별된 루트 정의 대신에 사용될 수 있다는 것을 표시하도록 한다. 이들 루트 타입은 항상 추상적이며 직접 실증될 수 없다.

[0460] 이들 타입의 정의는 모델 내의 그 실증을 제어하는 베이스 제한을 포함한다. 이들은 System.sdm에서 탐색될 수 있다.

[0461] 3.7.4 구체적인 객체 정의

[0462] 구체적인 객체 정의는 추상 객체 정의에 대한 구현을 제공한다. 구현은 객체 및 관계 멤버, 구현된 추상 정의의 세팅을 위한 값, 새로운 세팅 선언, 멤버들 간의 플로우, 및 멤버에 대한 제한으로 구성된다.

[0463] 구체적인 정의는 또한 네스트된 정의의 선언을 포함할 수 있다. 이들 정의는 포함 정의의 범위 내의 멤버에 대

하여 사용될 수 있고, 정의의 범위 밖의 제한에서 참조될 수 있다.

[0464] 3.7.4.1 베이스 구체적 객체 정의

[0465] 베이스 구체적 타입은 객체 정의, 상속 세팅 선언, 설계 데이터, 선택적 매니저 참조, 이름, 그것이 참여할 수 있는 관계에 대한 제한, 추상 정의의 세팅에 대한 값을 제공하는 능력 및 그 세팅과 그 멤버의 세팅 간의 플로우를 설명하는 능력을 확장한다. 구체적 정의는 그후 그것이 구현하는 추상 정의를 식별하는 능력을 부가하고 몇개의 선택적 속성은 정의의 결합 거동을 커스토마이즈하는 능력을 부가한다.

```
<xs:complexType name="ConcreteObjectDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:attribute name="Implements" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0466]

[0467]

속성/요소	설명
Implements	이러한 구체적인 타입을 구현하는 추상 타입을 식별

[0468] 3.7.4.2 객체 멤버

[0469] 객체 멤버는 추상 또는 구체적 객체 정의를 참조해야 한다. 객체 멤버는 어떤 경우 어레이에 대한 상부 및 하부 경계를 정의할 수 있는 인스턴스의 어레이를 표현할 수 있다. 객체 멤버가 참조 멤버이면, 객체를 실증하는 사용자는 멤버에 대한 인스턴스를 명시적으로 구성해야 한다. 객체 멤버가 참조 멤버가 아니면, 런타임은 외부 객체가 생성되는 것과 동시에 인스턴스를 생성할 것이다.

```
<xs:complexType name="ObjectMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="MinOccurs" type="xs:int" use="optional"/>
            <xs:attribute name="MaxOccurs" type="xs:int" use="optional"/>
            <xs:attribute name="IsReference" type="xs:boolean" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0470]

[0471]

속성/요소	설명
MinOccurs	이 멤버와 관련된 인스턴스의 번호에 대한 하한. 하한이 0이면, 수는 선택적이다. 디폴트는 1이다.
MaxOccurs	이 멤버와 관련된 인스턴스의 번호에 대한 상한. 1보다 커야한다. 디폴트는 1이다.
IsReference	이 값이 참이면, 멤버와 관련된 인스턴스는 오퍼레이터에 의해 명시적으로 생성되어야 하거나 또다른 타입에서 참조되어야 한다. 이 값이 거짓이면, 타입이 생성될 때 인스턴스가 생성된다.

[0472] sdm 모델에서, 우리는 부모가 구성될 때 생성되고 부모가 파괴될 때 파괴되는 멤버들과, 부모로부터 독립된 수명을 가질 수 있는 멤버들을 구별할 필요가 있다. 우리는 이 목적으로 IsReference 속성을 사용한다. new가 인스턴스를 생성하는 데 사용되는지에 기초하여 스택 기반 및 더미(heap) 기반 구성을 허용하는 C++ 선언과 유사하다. 멤버가 IsReference로서 마크되면, 명시적인 새로운 동작은 오퍼레이터의 부분에서 요구되어 인스턴스를 생성하고 그를 멤버와 관련시킨다.

[0473] 이것을 수행하는 다수의 이유가 있다.

[0474] 1. 오퍼레이터가 시스템을 구성할 때, 우리는 isReference 멤버를 구성하는 능력을 노출시킨다. 이것은 오퍼레이터 경험을 크게 간략화시킨다.

[0475] 2. 우리는 sdm 문서를 처리할 때, 문서의 인스턴스 공간이 구체적 정의 공간으로부터 변경될 수 있는 명백한 경계를 갖는다.

[0476] 3.7.4.3 관계 멤버(Relationship Member)

[0477] 관계 멤버는 생성될 때 객체 멤버들 사이에 존재할 관계를 식별한다. 관계 인스턴스는 오퍼레이터에 의해 명시적으로 생성되거나 런타임에 의해 암시적으로 생성된다. 전자의 예는 인스턴스들 사이의 호스팅 관계이고, 후자는 시스템 간의 통신 관계이다.

```
<xs:complexType name="RelationshipMember">
    <xs:complexContent>
        <xs:extension base="Member">
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0478]

[0479] 3.7.4.3.1 호스팅 멤버

[0480] 호스트 멤버는 2개의 객체 멤버 사이의 호스팅 관계를 선언하는 데 사용된다. 객체 멤버는 포함 정의를 갖는 직접 멤버(direct member)일 수도 있고, 멤버쉽 관계를 갖는 네스트된 멤버일 수도 있다. 참조된 멤버와 포함 정의 간의 멤버쉽 체인이 있어야 한다.

```
<xs:complexType name="HostingMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="GuestMember" type="Path" use="required"/>
            <xs:attribute name="HostMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0481]

[0482]

속성/요소	설명
GuestMember	관계의 게스트와 호환가능한 정의를 갖는 멤버를 식별. 멤버는 네스트될 수 있다.
HostMember	관계의 호스트와 호환가능한 정의를 갖는 멤버를 식별. 멤버는 네스트될 수 있다.

[0483] 3.7.4.3.2 통신 멤버

[0484] 통신 멤버는 정의의 중간 시스템 멤버의 엔드포인트 멤버들간의 통신 관계를 선언하는 데 사용된다.

```
<xs:complexType name="CommunicationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ClientMember" type="Path" use="required"/>
            <xs:attribute name="ServerMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0485]

[0486]

속성/요소	설명
ClientMember	관계의 클라이언트 정의와 호환가능한 정의를 갖는 엔드포인트 멤버를 식별. 엔드포인트 멤버는 정의의 중간 시스템 멤버의 멤버이어야 한다.
ServerMember	관계의 서버 정의와 호환가능한 정의를 갖는 엔드포인트 멤버를 식별. 엔드포인트 멤버는 정의의 중간 시스템 멤버의 멤버이어야 한다.

[0487] 3.7.4.3.3 포함 멤버

[0488] 포함 멤버는 타입 멤버가 타입에 의해 포함된 것을 선언하는 데 사용된다. 각각의 타입 멤버는 포함되거나 델리게이트될 수 있다. 포함 멤버는 관계의 "this" 포인터가 되도록 포함 관계의 부모 값을 자동적으로 설정한다.

```
<xs:complexType name="ContainmentMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ChildMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0489]

[0490]

속성/요소	설명
ChildMember	부모에 의해 포함될 인접 객체 멤버를 식별

[0491] 3.7.4.3.4 델리게이션 멤버

[0492] 델리게이션 멤버는 외부 타입에 대한 엔드포인트 정의 멤버와, 외부 타입의 인접 시스템 멤버에 대한 엔드포인트 정의 멤버 간의 델리게이션 관계를 셋업하는 데 사용된다.

```
<xs:complexType name="DelegationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ProxyMember" type="Path" use="required"/>
            <xs:attribute name="DelegateMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0493]

[0494]

속성/요소	설명
ProxyMember	프록시는 시스템에 대한 포함 관계를 갖지 않는 시스템에 대한 인접 엔드포인트 멤버를 식별한다. 멤버의 정의는 델리게이션 관계에 대한 프록시의 정의에 일치해야 한다.
DelegateMember	델리게이트는 타입의 인접 멤버에 대한 엔드포인트 멤버를 식별한다. 엔드포인트 멤버의 타입은 델리게이션 관계에 대한 델리게이트 타입에 일치해야 한다.

[0495] 3.7.4.3.5 참조 멤버

[0496] 참조 멤버는 외부 시스템의 2개의 인접 또는 네스트된 참조 관계를 셋업하는 데 사용된다.

```
<xs:complexType name="ReferenceMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="DependentMember" type="Path" use="required"/>
            <xs:attribute name="SourceMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
    </xs:complexType>
```

[0497]

[0498]

속성/요소	설명
dependentMember	소스 멤버에 의존하는 객체 멤버. 참조 관계 내의 종속 객체의 정의에 일치되어야 한다.
sourceMember	소스 객체 멤버. 참조 멤버 내의 소스 객체의 정의에 일치되어야 한다.

[0499] 3.7.4.4 엔드포인트 정의

[0500] 엔드포인트 정의는 네스트된 리소스 타입, 리소스 멤버 및 호스트, 포함 및 참조 관계 멤버를 선언하는 능력을 부가함으로써 베이스 객체 정의를 확장한다.

```
<xs:complexType name="EndpointDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0501]

[0502]

속성/요소	설명
ResourceDefinition	외부 타입 정의의 범위 내의 타입 멤버에 의해 사용될 수 있는 네스트된 리소스 정의
Resource	리소스 타입을 참조하는 리소스 멤버 선언
Hosting	호스팅 관계 멤버 선언
Containment	포함 관계 멤버 선언
Reference	참조 관계 멤버 선언

[0503] 3.7.4.5 서비스 정의

[0504] 시스템 타입은 네스트된 엔드포인트, 시스템 및 리소스 타입; 엔드포인트, 시스템 및 리소스 멤버, 및 호스트, 포함, 접속, 델리게이션 및 참조 관계에 대한 지원을 부가함으로써 베이스 타입을 확장한다.

```
<xs:complexType name="ServiceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="ServiceDefinition" type="ServiceDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Endpoint" type="EndpointMember"/>
                <xs:element name="Subsystem" type="SubSystem"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Connection" type="CommunicationMember"/>
                <xs:element name="Delegation" type="DelegationMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0505]

[0506]

속성/요소	설명
EndpointDefinition	외부 서비스 정의의 범위 내의 멤버에 의해 사용될 수 있는 네스트된 엔드포인트 정의
ResourceDefinition	외부 타입 정의의 범위 내의 타입 멤버에 의해 사용될 수 있는 네스트된 리소스 정의
SystemDefinition	외부 시스템 정의의 범위 내의 멤버에 의해 사용될 수 있는 네스트된 시스템 정의
Endpoint	엔드포인트 정의를 참조하는 엔드포인트 멤버 선언
Subsystem	시스템 정의를 참조하는 서브시스템 멤버 선언
Resource	리소스 정의를 참조하는 리소스 멤버 선언
Containment	포함 관계 멤버 선언
Hosting	호스팅 관계 멤버 선언
Connection	접속 관계 멤버 선언
Delegation	델리게이션 관계 멤버 선언
Reference	참조 관계 멤버 선언

[0507] 3.7.4.6 리소스 정의

[0508] 리소스 타입은 네스트된 리소스 타입 정의, 리소스 멤버, 및 호스트, 포함 및 참조 관계 멤버를 포함할 수 있다.

```
<xs:complexType name="ResourceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0509]

[0510]

속성/요소	설명
ResourceDefinition	외부 리소스 정의의 범위 내의 멤버에 의해 사용될 수 있는 네스트된 리소스 정의
Resource	리소스 정의를 참조하는 리소스 멤버 선언
Hosting	호스팅 관계 멤버 선언
Containment	포함 관계 멤버 선언
Reference	참조 관계 멤버 선언

[0511] 3.7.4.7 관계 규칙

[0512] 객체 정의의 특정 인스턴스에 대하여, 다음의 표는 인스턴스가 할 수 있는 역할의 각각과 관련된 카디널리티(cardinality)를 식별한다.

[0513] 3.7.4.7.1 시스템 규칙

[0514]

정의	역할	시스템	엔드포인트	리소스
System	Parent(0...*)	포함(Contains)	포함	포함
	Member(1...1)	ContainedBy	허용되지 않음	허용되지 않음
	Proxy(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Delegae(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Client(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Server(0..*)	허용되지 않음	허용되지 않음	허용되지 않음
	Guest(1..1)	HostedBy	HostedBy (??)	HostedBy
	Host(0..*)	호스트(Hosts)	호스트	호스트
	Source(0..*)	제공(Provides)	허용되지 않음	허용되지 않음
	Dependent(0..*)	소비(Consumes)	허용되지 않음	허용되지 않음

[0515] 3.7.4.7.2 엔드포인트 규칙

[0516]

	역할	시스템	엔드포인트	리소스

Endpoint	Parent(0...*)	허용되지 않음	허용되지 않음	포함
	Member(1...1)	ContainedBy	허용되지 않음	허용되지 않음
	Proxy(0...*)	허용되지 않음	DelegateTo	허용되지 않음
	Delegae(0...*)	허용되지 않음	구현(Implements)	허용되지 않음
	Client(0...*)	허용되지 않음	ConnectsTo	허용되지 않음
	Server(0..*)	허용되지 않음	ProvidesService	허용되지 않음
	Guest(1..1)	HostedBy	HostedBy	HostedBy
	Host(0..*)	호스트(Hosts)	호스트	호스트
	Source(0..*)	허용되지 않음	제공	제공
	Dependent(0..*)	허용되지 않음	소비	소비

[0517] **3.7.4.7.3 리소스 규칙**

[0518]

	역할	시스템	엔드포인트	리소스
Resource	Parent(0...*)	허용되지 않음	허용되지 않음	포함
	Member(1...1)	ContainedBy	ContainedBy	ContainedBy
	Proxy(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Delegae(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Client(0...*)	허용되지 않음	허용되지 않음	허용되지 않음
	Server(0..*)	허용되지 않음	허용되지 않음	허용되지 않음
	Guest(1..1)	HostedBy	HostedBy	HostedBy
	Host(0..*)	호스트(Hosts)	호스트	호스트
	Source(0..*)	허용되지 않음	제공	제공
	Dependent(0..*)	허용되지 않음	소비	소비

[0519] **3.7.4.7.4 주의**

[0520] 모든 인스턴스는 하나 이상의 호스팅 관계와 하나의 포함 관계를 정확히 참여시켜야 한다.

[0521] 이것은 다음을 의미한다.

[0522] A) 비참조 멤버가 포함 관계를 식별해야 한다.

[0523] B) 구성되기 위하여, 참조 멤버는 포함 관계를 식별해야 한다.

[0524] C) 포함 관계를 갖지 않는 참조 멤버는 다른 멤버를 델리게이트할 수 있을 뿐이다.

[0525] **3.8 관계**

[0526] 관계는 타입들 간의 가능한 상호 작용을 식별하는 데 사용된다. 관계는 바이너리이며 명령되고, 각각 관계 내에 참여할 수 있는 인스턴스의 타입을 식별한다. 관계는 또한 관계에 참여하는 인스턴스의 세팅을 포함할 수 있고 관계에 걸쳐 세팅 값을 플로우할 수 있다.

[0527] 다음은 타입 섹션에 설명된 웹서버에 대한 웹애플리케이션을 위한 가능한 호스팅 관계이다. 관계는 2개의 시스템의 보안 모델이 호환가능하다는 것을 검증하는 제한을 포함하고, vsite로부터 vdir로 서버 이름을 복사하는 세팅 플로우 멤버를 포함한다.

```
<AbstractHostingDefinition name="vsiteHostsVdir" guestType="vdir" hostType="vsite">

    <ObjectConstraint name="checkCompatibility" primaryRole="guest" primaryType="vdir">
        <Constraint name="constrainSecurityModel" type="SimpleOperatorComparison">
            <input name="LHS" path="host.securityModel"/>
            <settingValue name="operator">==</settingValue>
            <input name="RHS" path="guest.securityModel"/>
        </Constraint>
    </ ObjectConstraint >

    <flow type="copy" name="copyServerToVdir">
        <input name="source" path="host.server"/>
        <output name="destination" path="guest.server"/>
    </flow>

</ AbstractHostingDefinition >
```

[0528]

[0529] 관계는 관계에 참여할 타입 멤버를 식별하는 관계 멤버를 선언함으로써 사용된다.

```
<SystemDefinition name="testSystem" implements="ApplicationSpace">
    <resource name="myVdir" type="vdir" isReference="false"/>
    <resource name="myVsite" type="vsite" isReference="false"/>
    <hosting relationship="vsiteHostsVdir" guestMember="myVdir" hostMember="myVsite"/>
</ SystemDefinition >
```

[0530]

[0531] ### 3.8.1 관계 정의

[0532] 베이스 관계 정의는 정의에 객체 제한 및 플로우를 부가한다. 객체 제한은 이 관계의 인스턴스에 참여하는 객체 인스턴스에 대한 세팅 값에 관한 명령문이다. 예를 들어, DCOM 접속을 나타내는 통신 관계는 클라이언트 및 서버에 대한 보안 세팅이 호환가능한 것을 검사할 수 있다. 이 경우, 설계 프로세스의 부분으로서 쉽게 캡쳐될 수 있는 세팅들 간의 엄격한 관계가 있다; 관계에 대한 4 계승(4 factorial)개의 세팅 조합이 있지만, 더 적은 수의 유효 조합이 있을 수 있다.

[0533] 플로우는 하나의 인스턴스로부터 또다른 인스턴스로 값을 포워드하는 능력을 관계 개발자에게 제공한다. 이것은 객체 정의들이 그들의 가능한 상호작용으로부터 독립적으로 개발되도록 하고, 인스턴스가 특정 인스턴스를 완전히 설명하기 위하여 관계 그래프의 서브세트를 요구하기 보다는 정보에 대한 참조 포인트로서 독립하도록 한다.

[0534] 관계에 대한 이름은 관계를 포함하는 이름공간 내에서 고유해야 한다.

```
<xs:complexType name="RelationshipDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="Flow" type="FlowMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0535]

[0536]

속성/요소	설명
ObjectConstraint (group)	이 관계에 참여하는 인스턴스에 대한 제한. 섹션 3.5.3 참조.
Flow	이 관계에 참여하는 인스턴스들 간의 플로우

[0537] ### 3.8.2 추상 관계

[0538] 추상 관계는 2개의 추상 객체 정의 사이에서 정의된 관계이다. 추상 관계는 2개의 정의 사이의 가능한 상호작용을 나타낸다.

```
<xs:complexType name="AbstractRelationshipDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:attribute name="extends" type="QualifiedName" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0539]

[0540] 3.8.2.1 추상 통신 관계

[0541] 통신 관계는 엔드포인트 정의들 간의 가능한 통신 링크를 캡쳐하는 데 사용된다. 통신 관계는 독립적으로 배치된 소프트웨어 요소들 사이의 상호작용을 설명하는 데 사용된다. 통신 관계 스키마는 클라이언트 및 서버 엔드포인트 참조를 부가함으로써 베이스 관계 스키마를 확장한다.

```
<xs:complexType name="AbstractCommunicationDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ClientDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="ServerDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0542]

[0543]

속성/요소	설명
clientDefinition	통신 관계에 포함된 클라이언트 인스턴스의 정의
ServerDefinition	관계에 포함된 서버 인스턴스의 타입

[0544] 추상 타입 쌍의 다음의 조합은 통신 관계에 대하여 유효하다

[0545]

클라이언트 타입	서버 타입
엔드포인트	엔드포인트

[0546] 3.8.2.2 추상 호스팅 관계

[0547] 호스팅 관계는 게스트가 구성되기 위해 호스트를 요구한다는 사실을 캡쳐하는 데 사용된다. 게스트에 대한 가능한 호스트보다 더 많을 수 있기 때문에, 호스팅 관계는 또한 호스트에 대한 게스트의 구성을 담당한다. 객체의 인스턴스를 생성하기 위하여, 호스팅 관계는 게스트로부터 호환가능한 호스트로부터 존재해야 한다.

[0548] 예를 들어, 웹서비스 객체 정의와 IIS 객체 정의 사이에 호스팅 관계가 존재할 수 있다. 이 경우, MyWebservice 및 MyIIS가 웹서비스 및 IIS를 각각 구현하는 것으로 가정하면, 관계는 호스팅 관계에 대한 매니저를 사용하여 시스템 MyIIS의 인스턴스에 대한 시스템 MyWebservice의 인스턴스를 생성할 수 있다는 것을 표시한다. 우리는 시스템과 관계에 대하여 존재하는 제한을 평가할 때까지 관계를 생성할 수 있는지를 모른다.

```
<xs:complexType name="AbstractHostingDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="GuestDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="HostDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0549]

[0550]

속성/요소	설명
GuestDefinition	게스트 인스턴스의 정의를 식별
HostDefinition	호스트 인스턴스의 정의를 식별

[0551] 추상 정의 쌍의 다음의 조합은 호스팅 관계에 대하여 유효하다.

[0552]

게스트 타입	호스트 타입
엔드포인트	엔드포인트
리소스	리소스
리소스	시스템
시스템	리소스
시스템	시스템

[0553] 3.8.2.3 추상 포함 관계(Abstract Containment Relationship)

[0554] 2개의 추상 객체들 간의 포함 관계는 parentType에 기초한 구체적 타입이 memberType에 기초한 멤버를 포함할 수 있다는 사실을 캡쳐한다. 포함은 부모 인스턴스가 멤버 인스턴스의 수명을 제어할 수 있고 멤버 인스턴스에 거동을 델리케이트할 수 있다는 것을 암시한다.

```
<xs:complexType name="AbstractContainmentDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ParentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="MemberDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0555]

[0556]

속성/요소	설명
ParentDefinition	멤버를 포함하는 인스턴스의 정의를 식별
MemberDefinition	포함된 멤버가 될 인스턴스의 정의를 식별

[0557] 추상 정의 쌍의 다음의 조합은 포함 관계에 대하여 유효하다.

[0558]

부모 타입	멤버 타입
시스템	엔드포인트
시스템	리소스
시스템	시스템
엔드포인트	리소스
리소스	리소스

[0559] 3.8.2.4 추상 델리게이션 관계(Abstract Delegation Relationship)

[0560] 델리게이션은 외부 시스템으로부터 포함된 시스템으로 거동을 포워딩하는 데 사용된다. 이것을 수행하는 방법은 외부 시스템 상의 엔드포인트를 내부 시스템 상의 엔드포인트로 델리게이팅하는 것이다. 이것은 내부 시스템 상의 엔드포인트에 대하여 외부 시스템에 지시된 모든 상호작용을 효율적으로 포워딩한다. 델리게이션은 체인되어, 내부 시스템이 그 거동을 또다른 시스템에 델리케이팅하는 것을 허용할 수 있다.

[0561] 델리게이션 관계는 델리게이션에 참여할 수 있는 추상 엔드포인트 정의 쌍을 정의한다. 각각의 관계는 프록시로서 동작할 수 있는 추상 엔드포인트 정의와 거동을 델리케이팅할 수 있는 추상 엔드포인트 정의를 식별한다.

```
<xs:complexType name="AbstractDelegationDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ProxyDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="DelegateDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0562]

[0563]

속성/요소	설명
ProxyDefinition	내부 엔드포인트에 그 거동을 델리게이팅하는 외부 엔드포인트의 정의를 식별
DelegateDefinition	요구된 거동을 제공하는 내부 엔드포인트의 정의를 식별

[0564] 추상 타입 쌍의 다음의 조합은 델리게이션 관계에 대하여 유효하다.

[0565]

프록시 타입	델리게이트 타입
엔드포인트	엔드포인트

[0566] 우리는 리소스 및 시스템 델리게이션이 층들간의 결합을 지원하도록 할 수 있다. 예를 들어, IIS가 파일 시스템을 반드시 배치하지 않고서도 파일 시스템의 일부를 노출할 수 있게 한다.

[0567] **3.8.2.5 추상 참조 관계**

[0568] 우리는 호스팅 관계 종속성에 더하여 인스턴스들간의 강력한 의존성을 캡처하기 위하여 참조 관계를 사용한다. 이들 종속성은 배치동안의 구성 순서, 및 설치 및 갱신 동안의 시스템들 간의 플로우 파라미터를 제어하는 데 사용된다. 참조 관계는 강력한 종속성을 나타내므로, 우리는 참조 관계가 시스템 경계를 횡단하는 것을 허용할 수 없다. 이것은 하나의 시스템 내의 리소스가 또다른 시스템 내의 리소스에 대한 종속성을 가질 수 없다는 것을 의미한다. 이것은 시스템이 더이상 배치의 독립 유닛을 작성할 수 없게 한다. 시스템들간에 종속성이 존재하면, 우리는 통신 관계를 사용한다. 통신 관계는 시스템의 재설치를 필요로 하지 않고 시간에 따라 변경될 수 있다.

```
<xs:complexType name="AbstractReferenceDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="DependentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SourceDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0569]

[0570]

속성/요소	설명
DependentDefinition	소스 인스턴스에 의존하는 인스턴스의 정의
SourceDefinition	소스 인스턴스의 정의. 이 인스턴스는 종속성을 알 필요가 없다.

[0571] 추상 타입 쌍의 다음의 조합은 참조 관계에 대하여 유효하다.

[0572]

종속 타입	소스 타입
시스템	시스템
리소스	리소스

[0573] **3.8.3 암시적 베이스 관계(Implicit base relationship)**

[0574] 모든 추상 관계는 도 14에 도시된 바와 같이 베이스 관계 정의 중의 하나를 암시적으로 확장한다. 이들 정의는 관계 트리의 각각에 대한 루트를 형성한다. 이것을 수행함으로써, 우리는 제한 정의 내의 루트 정의를 참조할 수 있고 루트 타입으로부터 공통 타입 제한을 상속할 수 있다.

[0575] **3.8.4 구체적인 관계(Concrete relationship)**

[0576] 구체적 관계는 2개의 구체적 객체 정의 사이의 관계이다. 각각의 구체적 관계는 추상 관계를 구현해야 한다. 추상 관계는 구체적 객체 정의에 의해 직접적으로 또는 간접적으로(상속을 통해) 구현되는 추상 객체 정의의 일

치쌍 사이에 있어야 한다.

```
<xs:complexType name="ConcreteRelationship">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:attribute name="Implements" type="QualifiedName"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0577]

[0578] 3.8.4.1 호스팅 관계

[0579] 우리가 애플리케이션을 데이터센터에 배치할 때, 우리는 애플리케이션 내의 시스템에 대한 호스팅 관계를 모두 분석할 필요가 있다. 이것을 수행함으로써, 오퍼레이터는 요구된 호스팅 관계의 각각에 대한 호스팅 멤버를 생성할 필요가 있다. 오퍼레이터의 태스크를 간략화하고 개발자가 배치 프로세스를 안내하는 것을 허용하기 위하여, 개발자는 이를 대신하여 구체적 호스팅 관계를 생성할 수 있다. 구체적 호스팅 관계는 오퍼레이터가 애플리케이션을 배치할 때 단지 단일 호스팅 멤버를 선언할 필요가 있도록 호스팅 관계 멤버의 세트를 그룹화하는데 사용된다.

```
<xs:complexType name="HostingDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteRelationship">
            <xs:sequence>
                <xs:element name="Hosting" type="HostingMember" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="GuestDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="HostDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0580]

[0581]

속성/요소	설명
HostDefinition	이 호스팅 관계가 적용되는 게스트 구체적 객체 정의의 이름
GuestDefinition	이 호스팅 관계가 적용되는 호스트 구체적 관계 정의의 이름
Hosting	구체적 관계의 게스트 및 호스트 정의에 루트된 멤버를 참조하는 호스팅 관계 멤버의 리스트

[0582] 구체적 타입 쌍의 다음의 조합은 호스팅 관계에 대하여 유효하다.

[0583]

게스트 타입	호스트 타입
시스템	시스템

[0584] 게스트는 호스트 iff에 결합될 수 있다.

[0585] 게스트 내의 각각의 guestMember에 대하여

[0586] 호스트 내에 하나 이상의 hostMember가 존재한다

[0587] guestMember.Type은 hostMember.type를 갖는 호스팅 관계를 가지고,

[0588] guestMember.hostConstraints는 hostMember.settings을 유효화하고,

[0589] hostMember.guestConstraints는 guestMember.settings을 유효화하고,

[0590] guestMember의 각각의 멤버에 대하여, hostMember의 멤버에 대한 결합이 존재한다.

[0591] 예를 들어, 다음의 구체적 관계는 층 3 시스템(Bike)을 층 2 호스트(오퍼레이팅 시스템)에 결합한다. 이 경우, 우리는 "시스템 폴더"의 디폴트 값을 갖는 호스팅 관계에 대한 세팅을 정의한다. 우리는 층 3 애플리케이션의 시스템과 층 2 호스트의 시스템 간의 호스팅 관계를 정의하는 3개의 호스팅 멤버 중 하나로 이 세팅을 플로우시킨다.

```
<HostingDefinition name="DefaultBikePlacement" guestDefinition="Bike" hostDefinition="OperatingSystem:OperatingSystem">
    <settingDeclaration name="fileLocationRelativeToRoot" definition="xs:sting" access="readwrite" dynamic="false"/>
    <settingValue name="dirPath">systemFolder</settingValue>
    <flow name="copyPath" definition="copy">
        <input name="source" path="dirPath"/>
        <output name="destination" path="bikeExecutableHost.hostRelativePath"/>
    </flow>
    <hosting name="bikeExecutableHost" relationship="fileDirectoryHost"
                    guestMember="guest.bikeFile" hostMember="host.FileSystem"/>
    <hosting name="bikeEventKeyHost" relationship="registryKeyRegistryKeyHost"
                    guestMember="guest.bikeEventKey" hostMember="host.Registry.ApplicationEventKey"/>
    <hosting name="bikeSoftwareKeyHost" relationship="registryKeyRegistryKeyHost"
                    guestMember="guest.bikeSoftwareKey" hostMember="host.Registry.HKLM"/>
</HostingDefinition >
```

[0592]

[0593] ### 3.8.4.2 참조 관계

[0594] 우리는 2개의 구체적 타입들 간의 구체적 참조 관계를 사용하여 통신 관계를 포함하지 않는 시스템들간의 특정 종속성을 캡쳐할 수 있다. 예를 들어, 우리는 하나의 애플리케이션이 설치되기 위해서는 또다른 애플리케이션이 이미 존재하고 있어야 한다는 사실을 캡쳐할 수 있다.

```
<xs:complexType name="ReferenceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteRelationship">
            <xs:sequence>
                <xs:element name="Reference" type="HostingMember" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="DependentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SourceDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0595]

[0596]

속성/요소	설명
DependentDefinition	이 참조 관계가 적용되는 종속 구체적 객체 정의의 이름
SourceDefinition	이 참조 관계가 적용되는 소스 구체적 객체 정의의 이름
Reference	게스트 및 호스트 정의의 멤버를 참조하는 참조 관계 멤버의 리스트

[0597] 구체적 타입 쌍의 다음의 조합은 참조 관계에 대하여 유효하다.

[0598]

종속 타입	소스 타입
시스템	시스템
리소스	리소스
리소스	엔드포인트
엔드포인트	리소스

[0599] ## 3.9 객체 및 관계 제한

[0600] 우리는 객체 및 관계 제한을 사용하여 특정 관계에 사용될 때의 구체적 공간의 토폴로지를 정의하고 객체의 세팅을 제한한다.

[0601] 예를 들어, 추상 객체 정의(A)내에서, 우리는 이 추상 정의의 구현이 또다른 추상 객체 정의(B)의 하나의 인스턴스를 포함해야 한다는 것을 식별하기를 원한다. 하나 이상의 적절한 포함 관계가 이미 존재하는 것을 가정하면, 이것을 수행하기 위하여, 우리는 A 내에서 다음과 같이 보이는 관계 제한을 사용한다.

```
<RelationshipConstraint name="AContainsB"
                relationship="ContainmentDefinition"
                myRole="parent"
                targetType="B"
                minOccurs="1"
                maxOccurs="1"/>
```

[0602]

[0603] 제한은 A의 구현이 부모의 역할을 하고 관계의 다른 엔드에서의 타입이 타입 B인 포함 관계가 존재함을 식별한

다. 우리가 B의 구성을 통해 더 많이 제어하기를 원하면, 우리는 다음과 같이 타입 B의 세팅에 대한 제한을 부가할 수 있다.

```
<RelationshipConstraint name="AContainsB"
            relationship="ContainmentDefinition"
            myRole="parent"
            targetType="B"
            minOccurs="1"
            maxOccurs="1">

  <Constraint definition="simpleValueConstraint"
        name="BValueConstraint">

    <input name="LHS" path="member.name" />
    <settingValue name="operator">=</settingValue>
    <settingValue name="RHS">myPort</settingValue>

    </Constraint>
</RelationshipConstraint>
```

[0604]

[0605] 이 경우, 우리는 멤버의 이름이 스트링 "myPort"와 동일하도록 요구한 제한을 부가한다.

[0606] 우리는 또한 관계에 제한을 부가할 수 있고; 우리는 이들 객체 제한을 호출할 수 있다. 관계로부터, 우리는 관계에 참여한 객체를 제한한다. 관계 내의 각각의 역할에 대하여, 우리는 객체 정의를 식별할 수 있고 그 객체 정의에 세팅 제한을 부가할 수 있다. 그 관계로부터, 카디널리티는 항상 minOccurs=1 및 maxOccurs=1이므로, 이것은 제한 선언에 나타나지 않는다.

```
<ObjectConstraint name="allowedPair"
    primaryRole="host"
    primaryType="IIS"
    secondaryRole="guest"
    secondaryType="webApp"/>
```

[0607]

[0608] 마지막으로, 우리는 제한을 네스트할 수 있다. 이것은 제한들을 함께 체인할 수 있는 능력을 제공하고, 외부 제한은 내부 제한에 대한 컨텍스트를 세팅한다. 다음은, webApp를 특정 타입의 포함 엔드포인트로 제한하는 webapp 시스템을 호스트하는 IIS 시스템의 일례이다.

[0609] 이 경우, 우리는 하나 이상이 참이어야 하는 가능성의 세트를 지정하기 위하여 객체 제한의 그룹을 사용한다.

```
<AbstractSystemDefinition name="IIShost">
    <RelationshipConstraint name="WebAppHostConstraint" relationship="hostingRelationship" myRole="host"
targetType="webApp">
        <RelationshipConstraint name="WebAppContainsPort" relationship="containmentRelationship" myRole="parent"
targetType="portType">
            <ObjectConstraintGroup mode="oneTrue">
                <ObjectConstraint name="hasWebPort" primaryRole="member" primaryType="webPort"/>
                < ObjectConstraint name="hasSqlPort" primaryRole="member" primaryType="sqlPort"/>
            </ObjectConstraintGroup>
        </RelationshipConstraint >
    </RelationshipConstraint >
</ AbstractSystemDefinition >
```

[0610]

[0611] 네스트된 제한은 외부로부터 평가할 수 있는 경로를 형성한다. 경로 상의 각각의 제한은 현재의 인스턴스뿐만 아니라 경로 상의 이전의 인스턴스의 세팅을 액세스할 수 있다. 네스트된 제한의 평가는 마치 제한이 식별된 시스템내에 정의된 것처럼 수행된다.

[0612] foo의 관점으로부터, 다음의 2개의 시나리오는 동등하다. 제1 foo는 포함된 시스템 바 상의 네스트된 제한을 배치하고, 제2 foo에서는, 타입 바가 이미 제한을 포함한다.

[0613] 시나리오 1:

```
< AbstractSystemDefinition name="foo">
    < RelationshipConstraint name="containsBar" relationship="containment"
        myRole="parent" targetType="bar" minOccurs="1">
        ·· RelationshipConstraint name="containsX" relationship="containment"
            myRole="parent" targetType="X" minOccurs="1"/>
    </ RelationshipConstraint >
</ AbstractSystemDefinition >

< AbstractSystemDefinition name="bar"/>
```

[0614]

[0615] 시나리오 2:

```
< AbstractSystemDefinition name="foo">
    < RelationshipConstraint name="containsBar" relationship="containment"
        myRole="parent" targetType="bar" minOccurs="1"/>
</ AbstractSystemDefinition >

< AbstractSystemDefinition name="bar">
    < RelationshipConstraint name="containsX" relationship="containment"
        myRole="parent" targetType="X" minOccurs="1"/>
</ AbstractSystemDefinition >
```

[0616]

[0617] ### 3.9.1 제한 모델

[0618] 제한 모델에 2개의 부분, 즉 가드(guard)와 프리디케이트(predicates)가 있다. 우리는 가드를 사용하여 프리디케이트를 실행하는 컨텍스트를 정의한다. 예를 들어, 관계 내에서, 우리는 가드를 사용하여 프리디케이트를 실행하기를 원하는 타입의 특정 조합을 식별한다. 객체 내에서, 우리는 가드를 사용하여 다른 객체에 대한 관계의 세트를 식별한다.

[0619] 프리디케이트는 그 가드의 요구사항이 충족되면 실행된다. 우리는 프리디케이트의 2가지 형태 - 세팅 값을 유효화하는 세팅 제한 및 제한의 세트를 유효화하는 그룹 제한 - 를 갖는다: .

[0620] 우리는 가드 내에 가드를 네스트할 수 있고, 이 경우, 내부 가드는 외부 가드가 만족될 때에만 검사된다. 이것은 관계 구조의 검증을 지원하는 경로를 작성하도록 한다.

[0621] 가드와 그 프리디케이트의 조합은, 가드가 일치되어야 하고 프리디케이트가 참으로 평가하는 횟수를 나타내는 카디널리티를 가질 수 있다.

[0622] 더 형식적으로,

```
Guard :== ObjectConstraint(ObjectDefintion, ObjectDefintion ,required)
            { (Guard | predicate) * } |

          RelationshipConstraint(RelationshipDefinition,
                                 TargetObject,lBound,uBound)
            { (Guard | predicate) * }
```

[0623]

[0624] 가드는 ObjectConstraint 또는 RelationshipConstraint로서 정의된다. 객체 제한은 관계의 엔드에 관련된 2개의 객체 정의를 식별한다. 관계 제한은 관계 정의와 타겟 객체 정의를 식별한다. 객체 제한은 선택적이거나 요구될 수 있고, 관계 제한은 하한 및 상한을 갖는다. 이것은 관계가 2개의 타입을 식별할 수 있지만, 타입은 다수의 관계에 참여할 수 있다는 사실을 반영한다는 점에서 카디널리티와 다르다.

[0625]
```
Predicate :== SettingsConstraint(rule)  | group{ (guard)* }
```

[0626] 프리디케이트는 규칙을 포함하는 세팅 제한과 가드의 세트를 포함하는 그룹중의 하나이다. 프리디케이트는 가드의 컨텍스트 내에서 평가된다. 세팅 제한의 경우, 프리디케이트는 루트 가드의 소유자 및 각각의 네스트된 가드에 의해 식별된 컨텍스트로부터 세팅을 식별할 수 있다. 그룹은 하나 이상이 일치하고 참으로 평가되는 가드의 세트를 식별하는 데 사용된다.

[0627] 예:

[0628]
```
1. RelationshipConstraint(containmentRelationship,webapp,0,1){}
```

[0629] 이 예는 webapp에 대한 제한 관계가 있을 때마다 참으로 평가하는 가드를 나타낸다. 이 가드는 최대 한번 참으로 평가할 수 있다. 또한 일치는 사용자에게 에러를 리턴하도록 한다.

```
2. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      SettingsConstraint(webapp.name=2)
   }
```

[0630]

[0631] 이 예는 가드에 프리디케이트를 부가한다. 가드는 관계와 타겟 정의가 일치하고 세팅 제한이 참으로 평가될 때 단지 참으로 평가될 것이다. 관계와 타겟 정의가 일치하고 세팅 제한이 참이 아니면, 에러가 사용자에게 리턴

된다. 관계와 타겟 타입이 일치하고 세팅 제한이 한번 이상 참으로 평가되면, 에러가 사용자에게 리턴된다.

```
3. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      RelationshipConstraint(containmentRelationship,vdir,0,1)
   }
```

[0632]

[0633] 이 예에서, 우리는 가드 내에 가드를 네스트한다. 외부 가드가 참이면(제한을 포함하는 타입이 webapp를 포함하면), 우리는 외부 가드 내의 컨텍스트 내의 내부 가드를 평가한다. 이것은 내부 관계 제한이 webapp 인스턴스의 컨텍스트 내에서 평가되는 것을 의미한다. webApp가 0 또는 1의 vdirs를 포함하면, 내부 제한은 참을 리턴하고, 1보다 큰 vdir을 포함하면, 제한은 사용자에게 에러를 리턴한다.

```
4. ObjectConstraint(webapp,iis,0,1)
   {
      RelationshipConstraint(containmentRelationship,systemType,0,1)
      {
            TypeConstraint(webapp,vdir,0,1)
      }
   }
```

[0634]

[0635] 객체 제한의 컨텍스트는 주요 객체 정의(제1 객체 정의)이다. 이것은 관계 제한은 webapp의 컨텍스트 내에서 평가되는 것을 의미한다. 관계 제한은 2개의 가능한 컨텍스트를 정의하고, 제1 컨텍스트는 객체 제한을 위한 컨텍스이고 관계이고, 제2 컨텍스트는 관계 제한을 위한 컨텍스트인 타겟 객체 정의이다.

```
5. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      group
      {
         RelationshipConstraint(containmentRelationship,vdir,0,1)
         RelationshipConstraint(containmentRelationship,directory,0,1)
      }
   }
```

[0636]

[0637] 이 예에서, 우리는 그룹을 사용하여 Webapp의 컨텍스트 내에서 평가되는 2개의 관계 제한을 포함한다. 그룹은 관계들 중의 하나 이상이 발생되고 참을 리턴하지 않으면 에러를 발생시킨다. 이 경우, Webapp는 Vdir 또는 디렉토리를 포함해야 한다.

[0638] ### 3.9.2 베이스 제한

```
<xs:complexType name="Constraint">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="SimpleName"/>
</xs:complexType>
```

[0639]

[0640]

속성/요소	설명
Name	이 제한된 섹션의 이름
DesignData	이 제한에 관한 설계 표면 특정 정보

[0641] ### 3.9.3 객체 제한

[0642] 객체 제한은 관계의 역할 중 하나 또는 둘에 대한 제한을 설명한다. 제한은 실패하는 경우 제한의 식별을 돕기 위하여 이름을 가지며, 역할과 관련된 타입에서 타겟된 세팅 제한의 리스트를 포함하고, 역할과 관련된 정의로부터 도출된 객체가 되도록 인스턴스를 제한할 수 있다.

```
<xs:complexType name="ObjectConstraint">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="SettingsConstraint" type="ConstraintMember"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="PrimaryRole" type="RolesList" use="required"/>
            <xs:attribute name="PrimaryObject" type="QualifiedName" use="required"/>
            <xs:attribute name="SecondaryRole" type="RolesList" use="optional"/>
            <xs:attribute name="SecondaryObject" type="QualifiedName" use="optional"/>
            <xs:attribute name="Required" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0643]

[0644]

속성/요소	설명
SettingConstraint	이 제한의 컨텍스트에 대하여 적용된 세팅 제한의 리스트
RelationshipConstraint	네스트된 관계 제한. 관계가 주요 역할과 관련된 타입에 대하여 선언된 것처럼 평가된다.
RelationshipConstraintGroup	네스트된 관계 그룹 - 그룹 내의 관계는 1차 역할과 관련된 타입에 대하여 선언되는 것처럼 평가된다. 이전의 제한은 세팅 루트에 대한 공지된 이름이 된다.
PrimaryRole	이 제한이 목표로 하는 관계의 역할의 이름
PrimaryObject	1차 역할과 관련된 객체 정의의 이름
SecondaryRole	이 제한이 목표로 하는 관계에 대한 다른 역할의 이름
SecondaryObject	관계에 대한 2차 역할과 관련된 객체 정의의 이름. 이것은 2차 역할이 지정되면 요구된다.
Required	required가 참이면, 제한은 정의를 일치해야 하고 관계에 대한 역할에 대하여 선언하였다. required가 거짓이면, 가드는 사용에 있어서 타입을 일치하지 않는다. required는 관계가 타입의 특정 조합을 사용하도록 하는데 사용된다.

[0645] ### 3.9.4 객체 제한 그룹

[0646] 객체 제한 그룹은 객체 제한의 세트가 함께 그룹화되도록 하여 적어도 하나의 시멘틱을 사용하여 평가되도록 한다. 그룹은 객체 제한중의 하나 이상이 관계에 대한 객체와 일치하지 않으면 에러를 리턴하고 그 포함된 프리디케이트는 참으로 평가된다. 우리는 제한이 그룹의 직접 멤버이면 타입 제한에 대한 요구된 속성을 무시한다.

```
<xs:complexType name="ObjectConstraintGroup">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:sequence>
                <xs:element name="ObjectConstraint" type="ObjectConstraint" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0647]

[0648]

속성/요소	설명
ObjectConstraint	그룹 내에 정의된 타입 제한의 리스트

[0649] ### 3.9.5 관계 제한

[0650] 관계 제한은 객체가 참여할 수 있는 관계를 제한하는 데 사용된다. 관계 제한은 관계 정의, 관계의 다른 엔드에서의 인스턴스의 객체 정의 및 관계의 카디널리티를 식별한다. 에러 메시지 내에서 식별될 수 있도록 제한에 이름이 부여된다. 관계 제한의 바디는 관계와 관계의 다른 엔드에서의 인스턴스에 대한 프리디케이트를 포함한다.

[0651] 관계 제한은 다수의 목적으로 사용될 수 있다: 추가의 프리디케이트없이 카디널리티를 사용하여, 인스턴스에 제공되어야 하는 관계를 식별하여 프리디케이트로 정확하게 동작하는 데 사용되고, 이 객체가 상호 작용하려는 인

스턴스에 대한 구성의 세트를 좁히는데 사용될 수 있다.

```
<xs:complexType name="RelationshipConstraint">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="SettingsConstraint" type="ConstraintMember"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="Relationship" type="QualifiedName" use="required"/>
            <xs:attribute name="MyRole" type="RolesList" use="required"/>
            <xs:attribute name="TargetObject" type="QualifiedName" use="optional"/>
            <xs:attribute name="MinOccurs" type="MinOccurs" use="optional"/>
            <xs:attribute name="MaxOccurs" type="MaxOccurs" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0652]

[0653]

속성/요소	설명
SettingConstraint	관계 내의 세팅의 값 또는 관계의 다른 엔드에서의 객체에 대한 제한
RelationshipConstraint	타겟 객체(타겟 객체 정의는 지정되어야 한다)의 컨텍스트 내에서 평가되는 관계 제한. 이것은 제한을 타겟 객체에 부가하는 것과 동등하다.
RelationshipconstraintGroup	타겟 객체(타겟 객체는 지정되어야 한다)의 컨텍스트 내에서 평가되는 관계 그룹. 이것은 그룹을 타겟 객체에 부가하는 것과 동등하다.
ObjectConstraint	외부 제한에 의해 식별된 관계 정의의 부분인것처럼 평가된 네스트된 객체 제한.
OhjectConstraintGroup	외부 제한에 의해 식별된 관계 정의의 부분인 것처럼 평가된 네스트된 객체 제한 그룹.
Name	포함 정의의 범위 내의 제한에 대한 고유의 이름
Relationship	제한된 관계 정의의 이름
MyRole	이 객체 인스턴스가 이 관계에서 할 역할의 이름 - 이것은 관계 정의 내의 속성 이름에 대응하는 이름, 예를 들어, 클라이언트/서버, 게스트/호스트 등. 이것이 제공되지 않으면, 관계 내에 포함된 타입으로부터 역할을 추론한다.
TargetObject	관계의 다른 측에 나타날 수 있는 객체의 정의의 선택적 이름
MaxOccurs	이 객체의 시간 인스턴스의 최대 수가 명명된 관계 내의 정의된 역할의 참여자로서 식별될 수 있다. 이것이 제로이면, 타입은 명명된 관계에 참여하는 것을 명시적으로 금지한다.
MinOccurs	이 객체의 시간 인스턴스의 최소수가 명명된 관계 내의 정의된 역할에 참여자로서 식별될 수 있다.

[0654]

3.9.6 관계 제한 그룹

[0655] 관계 제한 그룹은 관계 제한의 세트가 함께 그룹화되어 하나 이상의 시멘틱을 갖는 프리디케이트로서 평가될 수 있도록 한다. 그룹은 포함된 관계 제한 중의 하나 이상이 관계 정의 및 타겟 객체를 일치시키지 않으면 에러를 리턴하고, 포함된 프리디케이트는 참을 리턴한다. 포함된 제한 내의 프리디케이트 중의 임의의 것이 에러를 리턴하면, 이들 에러는 사용자에게 전파된다. 포함된 관계 제한의 minOccurs 카디널리티는 무시되지만, maxOccurs 카디널리티가 위반되면, 에러가 사용자에게 리턴된다.

```
<xs:complexType name="RelationshipConstraintGroup">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:sequence>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0656]

[0657]

속성/요소	설명
relationshipConstraint	그룹 내에 정의된 관계 제한

[0658] ### 3.10 객체 매니저

[0659] 객체 매니저는 타입 및 관계가 고객 거동을 런타임 환경에 삽입하는 메카니즘이다. 매니저가 관리하는 각각의 타입을 지원할 수 있는 몇개의 역할이 있다: 타입의 설치에 참여할 수 있고, 타입의 CLR 표시를 제공할 수 있고, 타입들간의 결합이 어떻게 분석되는지에 대한 정책 결정에 포함될 수 있고 컴플렉스 제한과 플로우에 대한 구현을 제공할 수 있다.

[0660] 모든 객체 매니저 역할은 CLR을 통해 엔트리 포인트로서 강력하게 명명된 클래스로 노출한다. 객체 매니저는 sdm 내의 다른 타입과 동일한 방법으로 패키지되고 버저닝된다: 이들은 시스템 분산 유닛과 그들의 버전에 분산되고, 강력한 이름은 그들이 선언된 sdm 파일로부터 도출된다.

```
<xs:complexType name="Manager">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="AssemblyName" type="xs:string" use="required"/>
    <xs:attribute name="Version" type="FourPartVersionType" use="optional"/>
    <xs:attribute name="PublicKeyToken" type="PublicKeyTokenType" use="optional"/>
    <xs:attribute name="Culture" type="xs:string" use="optional"/>
    <xs:attribute name="Platform" type="xs:string" use="optional"/>
    <xs:attribute name="SourcePath" type="xs:string" use="optional"/>
</xs:complexType>
```

[0661]

[0662]

속성/요소	설명
Name	포함 sdm 파일의 범위 내의 이 매니저에 대한 고유의 이름
Description	매니저의 텍스트 설명
AssemblyName	어셈블리 이름
Version	어셈블리 버전
PublicKeyToken	어셈블리에 대한 공중 키 토큰
Culture	어셈블리의 컬쳐
Platform	어셈블리의 플랫폼
SourcePath	SDU 내의 어셈블리로의 경로

[0663] #### 3.10.1 역할

[0664] 객체 매니저는 지원하는 각각의 타입을 위한 하나 이상의 역할을 지원한다. 이들 역할은 다음을 포함한다.

[0665] a) 타입 또는 관계에 대한 제한 평가

[0666] b) 타입 또는 관계에 대한 플로우 평가

[0667] c) 타입에 대한 구성/해체/갱신 지원

[0668] d) 타입 또는 관계에 대한 세팅을 위한 객체 표시 노출

[0669] e) 타입 또는 관계에 대한 탐색 수행

[0670] f) 타입 또는 관계 주위의 설계 표면 특정 UI 지원

[0671] ### 3.11 SDM 문서 구조

[0672] sdm 문서는 강력한 식별, 관계의 세트에 대한 버저닝 및 로컬리제이션 정보, 객체 및 매니저를 제공한다.

```
<xs:element name="Sdm">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Information" type="Information" minOccurs="0"/>
            <xs:element name="Import" type="Import" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
            <xs:element name="SettingDefinitions" type="SettingDefinitions" minOccurs="0"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="AbstractEndpointDefinition" type="AbstractEndpointDefinition"/>
                <xs:element name="AbstractSystemDefinition" type="AbstractSystemDefinition"/>
                <xs:element name="AbstractResourceDefinition" type="AbstractResourceDefinition"/>
                <xs:element name="AbstractCommunicationDefinition" type="AbstractCommunicationDefinition"/>
                <xs:element name="AbstractHostingDefinition" type="AbstractHostingDefinition"/>
                <xs:element name="AbstractContainmentDefinition" type="AbstractContainmentDefinition"/>
                <xs:element name="AbstractDelegationDefinition" type="AbstractDelegationDefinition"/>
                <xs:element name="AbstractReferenceDefinition" type="AbstractReferenceDefinition"/>
                <xs:element name="ReferenceDefinition" type="ReferenceDefinition"/>
                <xs:element name="HostingDefinition" type="HostingDefinition"/>
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="ServiceDefinition" type="ServiceDefinition"/>
                <xs:element name="ConstraintDefinition" type="ConstraintDefinition"/>
                <xs:element name="FlowDefinition" type="FlowDefinition"/>
                <xs:element name="Manager" type="Manager"/>
            </xs:choice>
        </xs:sequence>
        <xs:attributeGroup ref="NamespaceIdentity"/>
        <xs:attribute name="documentLanguage" type="Culture"/>
    </xs:complexType>
</xs:element>
```

[0673]

[0674] ### 3.11.1 정보

[0675] SDM 문서의 정보 섹션은 인간이 판독가능한 정보를 포함하여 sdm 문서의 식별 및 관리를 지원한다.

```
<xs:complexType name="Information">
    <xs:annotation>
        <xs:documentation>Human readable information about the SDM Definition library.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="FriendlyName" type="xs:string" minOccurs="0"/>
        <xs:element name="CompanyName" type="xs:string" minOccurs="0"/>
        <xs:element name="Copyright" type="xs:string" minOccurs="0"/>
        <xs:element name="Trademark" type="xs:string" minOccurs="0"/>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="Comments" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

[0676]

[0677]

속성/요소	설명
FriendlyName	
CompanyName	
Copyright	
Trademark	
Description	
Comments	

[0678] ### 3.12 변경 요구

[0679] 도 15는 변경 요구의 일례를 나타낸다. 변경 요구가 SDM 런타임에 대한 변경의 세트를 식별한다. 런타임에 대한 모든 변경은 요구가 xml 포맷으로 구성되도록 하는 API를 통해 변경 요구를 사용하여 개시된다.

[0680] 초기 요구는 액션의 단일 그룹을 포함한다. 요구가 런타임에 의해 처리됨에 따라, 더많은 구조가 네스트된 그룹화를 통해 부가되고 더 많은 액션이 확장 및 플로우 프로세스의 결과로서 부가된다. 이 평가 프로세스를 통하고 타겟 머신에 대하여 실행될 준비가 된 변경 요구는 완전히 인가된 변경 요구라 불리운다. 더 많은 정보에 대하여 섹션 3.13을 참조하기 바란다.

[0681] 3.12.1 일관성 규칙(Consistency rules)

[0682] 액션이 SDM 인스턴스 공간 상에서 수행될 때, 우리는 액션이 SDM 내의 모든 인스턴스를 완료한 후 인스턴스 공간이 여전히 일관성 상태에 있는 것을 확인한다. 지속적 상태에 의해, 우리는 인스턴스에 적용되는 모든 제한이 여전히 유효하다는 것을 의미한다. 예를 들어, 서버로의 접속을 요구하는 클라이언트의 인스턴스를 생성하는 경우, 클라이언트를 생성하고 접속하는 데 사용되는 액션의 시퀀스가 완료할 때, 클라이언트와 서버 사이에 접속이 존재해야 한다.

[0683] 모델 일관성을 평가하기 위하여 사용된 제한은 액션 베이시스 또는 액션 세트의 결론에 대하여 평가될 수 있다. 우리는 동작 일관성과 처리 일관성의 2가지 형태의 일관성를 호출한다.

[0684] 트랜잭션이 완료된 후 객체가 일치하지 않으면, 우리는 사용자가 오프라인으로서 그 인스턴스를 명시적으로 마크하도록 한다. 인스턴스가 오프라인이면, 인스턴스에 적용되는 제한을 평가하지 않고 인스턴스는 다른 인스턴스의 관점으로부터 존재하는 것으로 나타나지 않는다. 이것은 이들 모든 인스턴스가 오프라인으로 마크될 수 있는 것을 의미한다. 오프라인은 부모로부터 자식으로 및 호스트로부터 게스트로 전파되므로, 시스템을 오프라인으로서 마크하는 것은 오프라인으로서 소유된 인스턴스내의 모든 것 및 호스트된 모든 인스턴스를 오프라인으로서 마크한다.

[0685] 3.13 모델 평가

[0686] 이 섹션에서, SDM 런타임의 범위 내의 SDM 모델의 거동을 설명한다.

[0687] 3.13.1 정의 공간

[0688] 정의 공간은 sdm 런타임으로 공지된 모든 정의를 포함한다. 도 16의 단계는 새로운 정의를 런타임으로 로딩하는 프로세스의 예를 정의한다. 이 프로세스는 또한 설계 표면이 sdm 문서를 유효화할 때 발생하는 컴파일 프로세스에 의해 공유된다.

[0689] 3.13.1.1 로드

[0690] sdm 문서는 sdu 또는 독립형 문서로서 런타임에 부여된다. 우리는 디스크로부터 파일을 로드하는 것을 시도한다.

[0691]

유효화 에러	설명
모르는 파일	우리는 특정 위치 내의 파일을 탐색할 수 없다.
액세스 거절	우리는 파일로의 액세스를 거절당한다.

[0692] 3.13.1.2 스키마 유효화

[0693] 제1 단계는 sdm 문서가 sdm 스키마에 일치되는 것을 유효화하는 것이다. 이 때, 우리는 모든 모르는 요소, 요구된 요소 또는 속성이 손실된 타입 또는 무효 데이터를 포함하는 타입에 대한 에러를 리턴할 것이다.

[0694]

유효화 에러	설명
무효 문서	문서는 무효 xml 신택스(syntax)를 갖는다 - 폐쇄되지 않은 노드, 하나 이상의 상부 레벨 노드 등.
모르는 요소	기대되지 않는 요소가 sdm 문서에서 탐색되었다.
모르는 속성	모르는 속성은 sdm 문서에서 탐색되었다.
무효 값	값이 스키마 유효화를 실패하였다(이것은 세팅 값 유효화를 포함하지 않는다)
손실된 속성	요구된 속성은 손실되었다.
손실된 요소	요구된 요소가 손실되었다.
무효 속성 조합	속성 또는 요소의 조합은 다음과 같이 무효한 것을 사용하였다. -minOccurs!=maxOccurs | minOccurs=0 byValue로

[0695]

우리는 모르는 요소 및 속성에 대하여 경고를 리턴하고 무시할 수 있다.

[0696] 3.13.1.3 세팅 값과 타입 분석

[0697] 타입 분석 페이즈에 있어서, 우리는 sdm 파일 내의 타입에 대한 모든 참조를 분석한다(인가된 이름은 스키마에 사용된다) 먼저, 우리는 문서의 범위내에 있는 모든 타입 참조가 유효하다는 것을 확인한다. 별칭을 포함하지 않는 모든 타입 참조가 있다. 그후, 우리는 모든 임포트 명령문을 분석하려고 시도한다. 우리가 임포트 명령문을 분석할 수 없으면, 우리는 이름공간 로드 에러를 생성하고, 우리가 임포트 명령문을 분석할 수 있으면, 우리는 이름공간 내의 타입을 위치지정하려고 시도한다. sdm 파일로부터 이름공간을 로드하려고 하면, 이름공간 분석 프로세스는 다른 에러를 발생시킬 수 있다.

[0698]

타입 분석 에러	설명
모르는 타입	타입은 로컬 또는 앨리어싱된 이름공간에서 탐색될 수 있다. 이것은 세팅, 시스템, 엔드포인트, 리소스, 관계, 제한 및 플로우에 대하여 발생될 것이다.
모르는 이름공간	이름공간은 탐색될 수 없다. 이름공간은 미리 로딩되었다.
버전 충돌	우리는 일치 버전 정보로 이름공간을 위치지정할 수 없다.
컬쳐 충돌	우리는 일치 컬쳐 정보로 이름공간을 위치지정할 수 없다.
타입의 무효 사용	이 상황에서의 무효 타입의 사용. 예를 들어, 관계 멤버 내의 세팅 타입 등.
무효 세팅 값	세팅 값은 그 타입의 유효화를 전달하는 것을 실패하였다.
불법 세팅 값	세팅 선언에 대한 액세스를 위반하거나 고정 값이 타입 또는 베이스 타입 내에서 미리 선언된 세팅 값이 제공되었다.

[0699] 3.13.1.4 경로 분석

[0700] 경로 분석 페이즈 동안, 우리는 문서 내에 정의된 세팅과 멤버로의 모든 경로를 분석하려고 시도한다. 분석되지 않은 타입을 갖는 세팅 또는 멤버를 참조하는 경로는 에러를 발생시키지 않는다.

[0701]

경로 분석 에러	설명
모르는 세팅	세팅 선언은 경로를 일치시키도록 탐색되지 않았다.
모르는 멤버	멤버 선언은 경로 내의 이름을 일치시키도록 탐색되지 않았다.
카디널리티 불일치	플로우 및 제한 명령문 내의 세팅 값으로의 경로는 1보다 큰 카디널리티를 갖는 관계 또는 멤버를 포함하였다.
타입 불일치	분석된 세팅 또는 멤버의 타입 및 변수의 선언된 타입은 매칭되지 않았다.
사용 불일치	경로의 선언된 목적 - 입력 또는 출력 - 값에 대한 고정 변경자 또는 세팅 선언에 대한 액세스 변경자를 위반하였다.
값이 없음	요구된 플로우 입력 경로는 디폴트 값을 갖지 않고 사용자가 값을 제공하도록 노출되지 않는 세팅을 참조하였다.

[0702]

경로 분석 경고	설명
런타임 경로 경고	추상 타입을 참조하는 isReference를 참조하는 경로는 런타임까지 검증할 수 없다.(우리는 사용자가 구체적 타입을 생성할 때까지 멤버가 존재하는 것을 검사할 수 없다.)
플로우 경고	플로우 경고 - 우리는 동시에 또는 더 일찍 정의된 플로우의 타겟인 세팅에 대한 값이 제공되면 경고를 발생시킬 것이다. 우리는 세팅이 고정되지 않는 한 세팅 값이 제공된 후 플로우가 정의되면 경고를 발생시키지 않는다.

[0703] 3.13.1.5 관계 참여

[0704] 타입 공간에서, 우리는 타입 선언이 관계 내의 그 멤버의 참여에 대하여 제한 중의 임의의 것을 위반하지 않은 것을 검사한다. 이것을 수행하기 위하여, 우리는 관련된 세팅 제한을 갖지 않는 관계 제한 및 모든 타입을 평가한다.

[0705]

타입 공간 제한 에러	설명
관계 타입 위반	관계 멤버는 특정 관계 및 관계에 의해 식별된 타입에 기초하여 양립할 수 없는 2개의 멤버를 식별한다. 예를 들어, vdirToVsite 호스팅 관계는 vdir 및 파일 사이에서 선언된다.
관계 사용 위반	관계는 2개의 멤버 사이에서 선언되지만, 이 컨텍스트 내의 관계의 사용은 허용되지 않는데, 그 이유는 멤버가 액세스가능하지 않기 때문이다. 예를 들어, 직접 포함 관계 내의 시스템 상이 아닌 2개의 엔드포인트들 간의 델리게이션 관계의 선언.
관계 제한 위반	관계는 관계 내의 제한에 기초하여 지원되지 않는 타입의 조합 사이에서 선언되었다. 예를 들어, 관계는 vsites 사이에서 선언될 수 있지만, vsite로부터 도출된 소정의 타입을 지원한다.

[0706]

타입 공간 제한 경고	설명
가능한 카디널리티 불일치	카운트업할 때 멤버에대한 maxOccurs는 관계 제한의 카디널리티를 위반한 관계의 세트를 발생시킨다.

[0707] **3.13.1.6 인스턴스 시뮬레이션**

[0708] 인스턴스 시뮬레이션에서, 우리는 실패한 것을 알아야 하는 제한을 식별할 수 있지만 사용자 입력에 기초하여 실패할 수 있거나 실패할 수 없는 제한을 플래그하도록 값을 플로우하고 제한은 평가하려고 시도한다. 이것을 수행하기 위하여 인스턴스 공간의 모델을 구성하고 이 인스턴스 공간에 기초한 제한 및 플로우를 평가한다. 플로우 또는 제한이 에러를 발생할 것으로 알면, 우리는 에러를 발생시키고, 에러를 발생시킬 수 있으면, 경고를 발생시킨다.

[0709] 우리는 모든 byReference 시스템에 대한 minOccurs 제한을 사용하여 인스턴스 공간 변경 요구를 작성한다. minOccurs가 0이면, 단일 인스턴스를 생성하고 그것을 선택적으로 마크한다. 우리는 표준 변경 요구를 사용함에 따라 동일한 확장 및 플로우 프로세스를 통해 변경 요구를 전달한다.

[0710]

시뮬레이션 경고	경고
선택적 시스템	시스템이 선택적이기 때문에, 런타임은 이 구성으로부터 발생할 수 있는 모든 에러를 완전히 결정할 수 없다.

[0711] 우리는 그후 완전히 정의된 입력 값을 갖는 모든 플로우를 평가한다. 입력 값이 고정되지 않고 사용자에 의해 변경될 수 있으면, 우리는 플로우의 출력을 임시로서 마크한다. 임시 입력은 그것을 소비하는 임의의 플로우 동작을 통해 체인할 것이다. 플로우가 완전한 입력 값을 갖지 않으며 사용자가 값을 제공할 수 있으면, 우리는 플로우의 모든 출력을 비정의(undefined)로 마크한다. 선택적 시스템으로부터의 플로우는 또한 임시 값을 발생시킨다.

[0712]

플로우 에러	설명
플로우 입력 비정의	값은 요구된 플로우 입력 값에 제공되지 않았다.

[0713] 일단 우리가 값을 플로우하면, 우리는 이들 값에 기초하여 제한을 평가한다. 임시 값을 실패한 제한은 경고로서 발생될 것이다; 경고는 제한이 비정의 값에 의해 평가될 수 없을 때 발생될 것이다.

[0714]

세팅 제한 에러	설명
세팅 입력 비정의	값은 요구된 제한 입력에 제공되지 않았다.
세팅 위반 제한	제한에 대한 하나 이상의 입력 세팅은 제한 위반을 발생시켰다.

[0715]

세팅 제한 경고	설명
세팅은 제한을 위반할 수 있다	디폴트에 기초한 입력 세트의 조합은 제한을 위반할 수 있다.

세팅 제한은 평가되지 않았다	제한은 배치 또는 사용 시간에 제공된 세팅에 의존하기 때문에 평가될 수 없다.

[0716] **3.13.2 인스턴스 공간**

[0717] 모델 평가 프로세스는 선언적 변경 요구를 제출함으로써 개시된다. 이 요구는 런타임 내의 인스턴스를 목표로 하는 생성, 갱신 또는 삭제 동작의 세트를 포함할 것이다. 우리는 도 17에 도시된 바와 같이 타겟 시스템 상의 요구된 변경을 규정하기 전에 일련의 파이프라인 스테이지를 통해 요구를 전달한다.

[0718] 다음의 섹션은 각각의 확장 단계의 책임을 약술한다.

[0719] **3.13.2.1 요구 제출**

[0720] 시스템에 대한 변경을 개시하기 위하여, 오퍼레이터 또는 프로세스는 변경 요구를 제출해야 한다. 변경 요구는 오퍼레이터가 런타임 내의 인스턴스들을 통해 수행하기를 원하는 액션의 세트를 포함하고; 이들 액션은 3개의 그룹, 즉 생성 액션, 갱신 액션 및 삭제 액션으로 나누어진다.

[0721] 요구는 그룹으로서 완료되거나 실패해야 하는 액션의 자동 세트로서 처리된다. 이것은 액션의 세트가 모델에 유효 변경을 발생시키는지를 평가할 때 제한 유효화 프로세스가 요구 내의 모든 액션을 고려하도록 한다.

[0722]

변경 요구 유효화 에러	설명
무효 문서	
모르는 요소/속성	모르는 요소 또는 속성은 xml 스키마에서 탐색되었다.

[0723] **3.13.2.1.1 타입 분석**

[0724] 타입 분석 페이즈에서, 우리는 변경 요구에서 참조되는 모든 타입 및 멤버를 분석한다. 변경 요구는 런타임에 의해 이미 로드된 것으로 가정될 것이다; 런타임은 변경 요구가 존재하지 않으면 로드/컴파일 액션을 초기화할 필요가 있다.

[0725] **3.13.2.1.2 경로 분석**

[0726] 경로 분석 페이즈 동안, 우리는 변경 요구 내의 액션을 생성함으로써 정의된 인스턴스 및 기존의 인스턴스에 대한 참조를 분석한다.

[0727] **3.13.2.2 확장**

[0728] 확장은 우리가 변경 요구를 취하고 요구를 실행하는 데 필요한 나머지 모든 액션을 차지하는 프로세스이다: 일반적으로 이들 액션은 타입 및 관계 인스턴스를 위한 구성 및 해체 액션이다. 이론적으로, 오퍼레이터는 인스턴스를 구성하거나 파괴하는데 요구되는 모든 액션을 위한 세부사항을 제공할 수 있지만, 우리는 이것을 요구하지 않는데, 그 이유는 변경 요구 작성 프로세스는 매우 복잡하기 때문이다. 대신에 우리는 이 프로세스를 자동화하려고 시도한다: 오퍼레이터는 byReference 멤버에 대한 액션을 식별함으로써 그들이 원하는 변경에 대한 키 정보를 제공한다; 우리는 네스트된 byReference 및 byValue 멤버 및 관계에 대한 액션의 나머지를 채운다.

[0729] **3.13.2.2.1 값 멤버**

[0730] 확장 스테이지 동안, 우리는 모든 비참조(non-reference) 타입 멤버를 식별한다. 우리는 이들 멤버의 카디널리티를 알고 요구된 모든 파라미터를 알고, 각각의 멤버에 대하여, 우리는 그 부모가 생성된 그 멤버에 대한 변경 요구에 생성 요구를 부가한다. 변경 요구가 해체 동작을 포함하면, 우리는 모든 그들의 포함된 인스턴스에 대한 해체 동작을 부가한다.

[0731] **3.13.2.2.2 참조 멤버 확장(탐색)**

[0732] 일반적으로, 참조 멤버는 더 많은 정보를 요구하여 값 멤버를 구성한다. 그들의 카디널리티는 종종 정의되지 않고 이들은 인스턴스에 대하여 구성되도록 값을 요구하는 배치 시간 세팅을 가질 수 있다. byReference 멤버를 확장하는 프로세스는 제공하는 위치에 있는 런타임보다 더 많은 인스턴스에 대한 정보를 요구할 수 있다.

우리가 이 정보를 얻는 프로세스는 탐색이라 한다.

[0733] 탐색의 프로세스는 구성 또는 갱신 액션의 부분으로서 참조 타입 멤버를 차지할 것이다. 탐색을 지원하는 객체 매니저를 갖는 참조 멤버만이 이 프로세스에 참여할 것이다.

[0734] 새로운 인스턴스가 발견될 때, 우리는 먼저 인스턴스 특정 키 값을 이용하여 인스턴스가 SDM 데이터베이스에 이미 존재하지 않는 것을 검사한다. 일단 우리가 그것이 새로운 인스턴스인 것을 알면, 우리는 우리가 탐색한 멤버의 타입에 따라 인스턴스를 분류한다. 인스턴스가 멤버와 일치하지 않거나 모호하게 일치되면, 우리는 멤버를 참조 블랭크로 내버려 두고 인스턴스를 오프라인 및 불완전으로 마크한다.

[0735] **3.13.2.2.3 관계 확장**

[0736] 일단 구성될 모든 타입 인스턴스를 알면, 타입 인스턴스를 함께 결합하는 관계 인스턴스를 생성한다. 타입 인스턴스가 파괴되면, 타입 인스턴스를 참조하는 모든 관계 인스턴스를 제거한다.

[0737] 관계를 생성하기 위하여, 인스턴스들 사이에 존재해야 하는 관계의 구성을 식별하기 위하여 멤버 공간으로 돌린다. 타입이 1보다 큰 카디널리티를 가지면, 우리는 관계의 토폴로지를 추론한다. 섹션 XX에서 이것을 수행하는 방법을 설명할 것이다.

[0738] **3.13.2.3 플로우**

[0739] 플로우 스테이지 동안, 모든 관계 인스턴스에 걸친 플로우를 평가한다. 이 스테이지는 갱신 요구를 변경된 파라미터 플로우에 의해 영향을 받은 인스턴스에 대한 변경 요구에 부가할 수 있다.

[0740] 플로우는 변경 요구의 결과로서 갱신된 세팅을 갖는 인스턴스의 세트를 결정함으로써 평가된다. 이들 각각에 대하여, 변경된 세팅에 의존하는 임의의 아웃고잉 세팅 플로우는 평가되고 타겟 노드는 변경된 인스턴스의 세트에 부가된다. 프로세스는 그 세트가 비거나 세트가 사이클을 포함할 때까지 계속된다.

[0741]

에러/경고	설명
종료되지 않은 플로우	

[0742] **3.13.2.4 이중 검출**

[0743] 이중 검출의 프로세스는 sdm 데이터 저장부에 이미 존재하는 인스턴스에 대하여 확장된 인스턴스와 일치된다. 예를 들어, 또다른 애플리케이션이 공유된 파일을 설치하는지를 검출한다. 인스턴스가 이미 존재하는 것을 검출하면, 기존의 인스턴스의 버전에 의존하여 몇개의 액션중 하나의 액션을 취한다.

[0744] a) 설치를 실패할 수 있다.

[0745] b) 인스턴스의 카운트를 참조할 수 있다.

[0746] c) 인스턴스를 업그레이드할 수 있다.

[0747] d) 차례로 설치할 수 있다.

[0748] **3.13.2.5 제한 평가**

[0749] 제한 평가 페이즈 동안, 변경 요구가 처리된 후 모델 내의 모든 제한이 여전히 유효한 것을 검사한다.

[0750]

v1에 대하여, 제한의 범위를 결정하는 것이 어려울 수 있기 때문에 제한을 갖는 그래프 내의 모든 노드를 방문해야 한다.(인스턴스 공간을 제거할 수 있는 방법으로 멤버 공간을 태그할 수 있다)

[0751] **3.13.2.6 요구 순서화**

[0752] 우리는 액션의 완전한 리스트를 가지며, 우리는 시스템간의 관계를 사용하여 유효 변경 순서화를 결정할 수 있다.

[0753] **3.13.2.7 실행**

[0754] 우리는 머신 특정된 액션의 순서 세트의 서브세트를 분산한다. 우리는 이들 머신 특정 세트의 머신 동기화에

대하여 지원해야 한다.

[0755] **3.13.2.8 요구 리턴**

[0756] 변경은 영향을 받은 호스팅 관계에 기초하여 변경 요구를 분산가능 부분으로 분류함으로써 수행된다. 모든 부분이 완료되면(또는 실패하면) 그 결과는 런타임에서 대조되고 사용자에게 요약이 리턴된다.

[0757] **3.13.3 깊이의 확장**

[0758] 이 섹션에서, 타입 및 관계에 대한 확장 프로세스에 대한 세부사항을 설명한다.

[0759] **3.13.3.1 참조 번호 확장 (탐색)**

[0760] 호스팅 관계가 타입의 새로운 인스턴스를 구성하는 것과 동일한 방법으로, 우리는 호스팅 관계를 사용하여 기존의 타입 인스턴스를 탐색한다. 타입 인스턴스가 호스트 상에 표시되는 방법을 알고 있는 것처럼, 호스팅 관계는 유일하게 배치되어 이것을 수행한다.

[0761] 참조 멤버가 탐색을 위해 마크될 때, 호스팅 관계가 탐색을 지원하는지를 검사한다. 지원하면, 우리는 호스트 인스턴스를 관계에 전달하고, 그것이 호스트 상에서 탐색한 게스트 인스턴스에 대한 구성 액션을 리턴하도록 요청한다.

[0762] 우리는 인스턴스가 더이상 존재하지 않는 것을 탐색하기 위하여 검증을 사용한다. 이것은 다시 호스팅 관계를 사용하여 호스트 상의 게스트의 존재를 검증한다. 게스트가 더이상 존재하지 않으면, 호스팅 관계는 해체 액션을 변경 요구에 부가한다.

[0763] **3.13.3.2 비참조 멤버 확장**

[0764] 런타임은 변경 요구 내의 구성 또는 해체를 위해 이미 식별된 타입의 각각의 비참조 멤버에 대한 구성 또는 해체를 단순히 부가함으로써 모든 비참조 멤버 확장을 핸들링한다.

[0765] **3.13.3.3 통신 관계 확장**

[0766] 오퍼레이터가 2개의 타입 멤버들 간에 통신 관계 멤버가 존재하는 통신 관계의 인스턴스를 지정하지 않으면, 우리는 멤버들 간의 완전히 접속된 메쉬(mesh)를 추측함으로써 통신 관계를 확장한다.

[0767] 캡쳐할 수 없는 중요한 토폴로지가 있으면 접속성 제한은 늦추어지지만, 늦추는 것은 배치 프로세스를 복잡하게 한다. 예를 들어, 토폴로지를 더 늦추면, 와이어를 생성하고 관리하는 방법을 제공하고, 임의의 변경에 대한 경로 검사를 수행하고, 토폴로지를 배치를 생성하는 오퍼레이터에게 노출시킬 수 있다.

[0768] 이것이 무엇을 의미하는가? 2개의 멤버가 멤버 공간 내에 접속되면, 각각의 멤버의 모든 인스턴스가 서로 관찰될 수 있다. 다음의 2개의 멤버가 주어지면, 도 18에 도시된 바와같이, 멤버의 카디널리티에 의해 인스턴스 공간 토폴로지가 제한된다. 2가지 예의 멤버가 1800에 도시된다. 1802에서, 2개의 인스턴스의 최대 사이의 단순한 포인트 대 포인트 관계가 도시된다. 1804에서, 접속의 팬 아웃(fan out)이 도시된다. 예는 서버의 세트에 걸쳐 밸런스 요구를 로드할 수 있는 클라이언트일 수 있다. 1806에서, 접속의 팬 인(fan in)이 도시된다. 예는 단일 서버를 공유하는 클라이언트의 그룹일 수 있다. 1808에서, 클라이언트의 세트가 서버의 세트를 공유하는 상기 경우의 조합이 도시된다.

[0769] 통신 링크를 구성할 때, 델리게이트 엔드포인트는 투과되어 델리게이트 엔드포인트가 제거되면 존재하는 모든 통신 관계를 일치시키는 접속으로 종료하도록 한다. 도 19는 A, B 및 C의 인스턴스들간의 접속을 고려하는 한 동등한 2개의 구조(1902, 1904)를 도시한다.

[0770] **3.13.3.4 호스팅 관계 확장**

[0771] 호스팅 관계가 모호한 경우, 우리는 호스팅의 관계의 매니저 또는 오퍼레이터에게 정확한 토폴로지를 결정할 것을 요구한다.

[0772] 호스팅 관계가 확장을 지원하면, 호스트 및 게스트의 세트를 관계 매니저에 전달하고 매니저에게 정확한 구성 액션을 리턴하도록 요청한다. 매니저가 확장을 지원하지 않으면, 변경 요구를 오퍼레이터에게 리턴하여 더 많

은 정보를 제공할 수 있다.

[0773] **3.13.3.5 참조 관계 확장**

[0774] **3.13.3.6 제한 관계 확장**

[0775] 제한 관계는 런타임이 항상 적절한 구성 액션을 변경 요구에 부가할 수 있도록 모호하지 않다.

[0776] **3.13.3.7 델리게이션 관계 확장**

[0777] 확장을 위해, 델리게이션 관계는 통신 관계와 동일한 규칙을 따른다.

[0778] **3.13.4 플로우**

[0779] **3.13.5 실행**

[0780] **3.14 SDM 인스턴스 공간**

[0781] 다음의 섹션은 sdm 런타임의 인스턴스 공간에 대한 객체 모델을 정의한다. 인스턴스 공간은 sdm에 의해 모델링된 시스템의 구성에 대한 변경을 추적하는 데 사용된다.

[0782] 도 20은 인스턴스 공간의 개요을 제공하는 일례의 UML 다이어그램을 나타낸다. 박스(2002, 2004, 2006, 2008)는 이 문서의 다른 섹션에서 정의된 타입을 나타낸다.

[0783] 인스턴스 공간은 변경 요구에 의해 개시된 버저닝된 변경에 대하여 구조화된다. 각각의 인스턴스는 운영 인스턴스에 작성된 미소한 변경을 나타내는 버전의 선형 시리즈를 가질 수 있다. 미래의 버전은 또한 운영 시스템에 전파되기 전에 런타임에 존재할 수 있다.

[0784] SDM 모델의 이 버전에 대하여, 주어진 인스턴스에 대하여 선형 변경을 허용한다. 미래에, 버전 브랜치를 허용할 수 있고 버전 분석 모델을 도입한다. 이것은 특정 인스턴스에 대하여 하나 이상의 변경이 나타나도록 한다.

[0785] 선형 버저닝을 허용하므로, 이전의 변경을 작성하는 일련의 변경 요구를 로드할 수 있다. 이것은 롤링 업그레이드 등의 프로세스 동안 취해질 수 있는 액션의 시퀀스의 이전 유효화를 지원한다.

[0786] **3.14.1 SDM 인스턴스**

[0787] 모든 인스턴스는 sdm 인스터로부터 도출된다. 인스턴스는 세팅 스키마를 위한 값을 정의하는 요소와 인스턴스의 정의에 대한 멤버와 일치하는 멤버의 리스트를 공유한다. 또한 인스턴스는 인스턴스에 대한 고유의 식별자를 정의하는 속성의 세트, 인스턴스에 대한 버전 번호, 이 버전이 시스템의 운영 상태를 나타내는지를 표시하는 플래그 및 인스턴스에 대한 이름을 공유한다.

```
<xs:complexType name="sdmInstance">
    <xs:sequence>
        <xs:element name="settingValues" type="settingValues" minOccurs="0"/>
        <xs:element name="member" type="member" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="instanceID" use="required"/>
    <xs:attribute name="version" type="xs:int" use="required"/>
    <xs:attribute name="isCurrent" type="xs:boolean" use="required"/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="incomplete" type="xs:boolean" use="required"/>
</xs:complexType>
```

[0788]

[0789]

속성/요소	설명
settingsValues	모델링된 시스템 객체의 원하는 상태를 정의하는 세팅 값의 리스트. 이 리스트는 인스턴스를 배치할 때 또는 관련된 인스턴스로부터의 플로우를 통할 때, 오퍼레이터, 정의 또는 멤버에 대한 개발자에 의해 정의된 모든 값을 포함한다.
member	이것은 정의에 대한 멤버를 표시하는 멤버의 리스트이다. 각각의 멤버는 멤버에 할당된 인스턴스를 식별한다.
id	글로벌 범위에서 고유한 인스턴스에 대한 식별자(분산된 런타임을 지원하기 위하여)
version	버전 번호는 인스턴스에 대한 변경으로 선형적으로 증가한다.
isCurrent	이것은 이 버전이 시스템의 운영 상태를 표시하는지를 표시하는 플래그이다. 운영 시스템에 전파되지 않은 갱신을 나타내는 후자의 버전이 존재할 수 있다.
name	그 포함 멤버의 배경 내의 인스턴스를 위한 고유 이름(데릴게이트된 멤버의 관점에서 고유하지 않을 수 있다)

[0790] 3.14.2 멤버

[0791] 멤버는 참조된 멤버의 세트인 인스턴스의 멤버를 관련시키는 데 사용된다. 인스턴스의 멤버는 인스턴스의 정의에 의해 정의된다. 참조된 인스턴스는 멤버가 델리게이트되는 인스턴스 또는 멤버에 대하여 생성된 인스턴스이다. 멤버는 하나 이상의 참조된 인스턴스일 수 있는 어레이를 나타낼 수 있다.

```
<xs:complexType name="member">
    <xs:sequence>
        <xs:element name="instance" type="instanceRef" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="memberDeclaration" type="qualifiedName" use="optional"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

[0792]

[0793]

속성/요소	설명
instance	이 멤버에 의해 참조된 인스턴스
memberDeclaration	관련된 정의에 대한 이 멤버의 선언
name	정의에 대한 관련된 멤버의 이름

[0794] 3.14.3 변경

[0795] 변경은 인스턴스 상태에 대한 변경을 표시한다. 변경은 변경 요구를 영향을 받은 인스턴스의 세트와 관련시킨다. 변경이 실행되면, 변경은 또한 변경의 상태(섹션 XXX 참조)와 변경 응답을 식별한다.

```
<xs:complexType name="change">
    <xs:sequence>
        <xs:element name="instance" type="instanceVersionRef" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="changeID" use="required"/>
    <xs:attribute name="status" type="changeStatus" use="required"/>
    <xs:attribute name="changeRequest" type="qualifiedName" use="optional"/>
    <xs:attribute name="changeResponse" type="qualifiedName" use="required"/>
</xs:complexType>
```

[0796]

[0797]

속성/요소	설명
instance	관련된 변경 요구의 결과로서 생성한 인스턴스 버전의 리스트
id	이 변경을 위한 고유의 식별자(런타임에서는 적어도 고유하다)
status	이 변경의 현재 상태를 식별하는 열거
changeRequest	이 변경을 생성하는 데 사용된 변경 요구에 대한 리스트
changeResponse	변경의 실행으로부터 리턴된 결과에 대한 리스트

[0798] 3.14.3.1 변경 상태

[0799] 변경 요구는 다음의 상태 중의 하나에 있을 수 있다.

[0800] • notstarted-변경 요구에 대하여 실행이 시도되지 않았다는 것을 표시

[0801] • inProgress-현재 실행되는 것을 표시

[0802] • complete-변경 요구가 성공적으로 완료되었다는 것을 표시

[0803] • failed-변경 요구가 실패하였고 변경이 불완전한 상태에 있는 것을 표시

[0804] • rolledBack-실패한 변경 요구가 성공적으로 롤백되었다는 것을 표시

```
<xs:simpleType name="changeStatus">
    <xs:restriction base="xs:string">
        <xs:enumeration value="notStarted"/>
        <xs:enumeration value="inProgress"/>
        <xs:enumeration value="completed"/>
        <xs:enumeration value="failed"/>
        <xs:enumeration value="rolledBack"/>
    </xs:restriction>
</xs:simpleType>
```

[0805]

[0806] ### 3.14.4 구체적 객체 인스턴스

[0807] 구체적 객체 인스턴스는 타입 속성에 의해 식별된 구체적 타입의 인스턴스를 표시한다. 인스턴스에 대한 실세계 표시가 있을 수 있으므로, 인스턴스가 실세계 부본과 동기하는지를 추적할 필요가 있다. 우리는 인스턴스가 이 변경의 결과로서 온라인일 지를 알기를 원한다. 온라인 인스턴스는 그 모든 제한에 대하여 유효해야 한다. 오프라인 인스턴스는 참여하는 통신 관계의 다른 참여자에게 보여지도록 나타나지 않는다. 인스턴스가 불완전하면, 변경 요구는 인스턴스가 온라인될 수 있기 전에 요구된다.

```
<xs:complexType name="ObjectInstance">
    <xs:complexContent>
        <xs:extension base="sdmInstance">
            <xs:attribute name="inSync" type="xs:boolean" use="required"/>
            <xs:attribute name="online" type="xs:boolean" use="required"/>
            <xs:attribute name="type" type="qualifiedName" use="required"/>
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

[0808]

[0809]

속성/요소	설명
inSync	이것은 인스턴스 상의 세팅이 그 실세계 부분 상의 세팅과 일치하는지를 표시한다.
online	이것은 실세계 부본이 온라인 및 액티브이어야 하는지를 표시한다. 임의의 제한이 만족되지 않으면, 인스턴스는 온라인 상태에 놓일 수 없다. 오프라인 인스턴스는 그것을 참조하는 통신 관계 내의 다른 참여자에게 보여질 수 없다(플로우는 실행되지 않을 것인가?)
incomplete	이 플래그는 인스턴스의 이 버전으로부터 요구된 정보가 손실되는 것을 표시한다. 이 상황은 인스턴스에 의해 요구되는 모든 정보를 식별할 수 없는 탐색 프로세스의 결과 또는 모든 요구된 정보를 공급하지 않는 변경 요구의 결과로서 발생할 수 있다.
type	인스턴스의 타입에 대한 참조

[0810] ### 3.14.5 관계 인스턴스

[0811] 관계 인스턴스는 식별된 관계 타입의 인스턴스를 표시한다. 관계가 직접 실세계 표시가 가지지 않으면, 관계가 동기하거나 온라인인지에 대한 정보를 유지해야 한다. 또한, 관계가 비교적 간단하면, 관계가 그들의 제한을 실패할 수 있어도, 관계가 불완전한 것을 기대하지 않는다.

```
<xs:complexType name="relationshipInstance">
    <xs:complexContent>
        <xs:extension base="sdmInstance">
            <xs:attribute name="relationship" type="qualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0812]

[0813]

속성/요소	설명
relationship	이 인스턴스와 관련된 관계 타입

[0814] #### 3.14.5.1 포함 인스턴스

[0815] 이것은 포함 관계의 인스턴스를 나타낸다.

```
<xs:complexType name="containmentInstance">
        <xs:complexContent>
            <xs:extension base="relationshipInstance">
                <xs:attribute name="parentInstance" type="instanceID" use="required"/>
                <xs:attribute name="childInstance" type="instanceID" use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

[0816]

[0817]

속성/요소	설명
parentInstance	관계에 참여한 부모 인스턴스를 식별
childInstance	관계에 참여한 자식 인스턴스를 식별

[0818] 3.14.5.2 통신 인스턴스

[0819] 이것은 통신 관계의 인스턴스를 표시한다

```
<xs:complexType name="communicationInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="clientInstance" type="instanceID" use="required"/>
            <xs:attribute name="serverInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0820]

[0821]

속성/요소	설명
clientInstance	관계에 참여한 클라이언트 인스턴스를 식별
serverInstance	관계에 참여한 서버 인스턴스를 식별

[0822] 3.14.5.3 정의 인스턴스

[0823] 이것은 델리게이션 관계의 인스턴스를 나타낸다.

```
<xs:complexType name="delegationInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="proxyInstance" type="instanceID" use="required"/>
            <xs:attribute name="delegateInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0824]

[0825]

속성/요소	설명
proxyInstance	관계에 참여한 프록시 인스턴스를 식별
delegateInstance	관계에 참여한 델리게이트 인스턴스를 식별

[0826] 3.14.5.4 호스팅 인스턴스

[0827] 이것은 호스팅 관계의 인스턴스를 나타낸다.

```
<xs:complexType name="hostingInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="guestInstance" type="instanceID" use="required"/>
            <xs:attribute name="hostInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0828]

[0829]

속성/요소	설명
guestInstance	관계에 참여한 게스트 인스턴스를 식별
hostInstance	관계에 참여한 호스트 인스턴스를 식별

[0830] 3.14.5.5 참조 인스턴스

[0831] 이것은 참조 관계의 인스턴스를 나타낸다.

```
<xs:complexType name="referenceInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="sourceInstance" type="instanceID" use="required"/>
            <xs:attribute name="dependentInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

[0832]

[0833]

속성/요소	설명
sourceInstance	관계에 참여한 소스 인스턴스를 식별
dependentInstance	관계에 참여한 종속 인스턴스를 식별

[0834] 3.14.4.6 인스턴스

[0835] 인스턴스는 sdmInstance 파일내에 존재할 수 있는 인스턴스 요소의 세트를 표시한다.

```
<xs:group name="instances">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="SystemInstance" type="concreteTypeInstance"/>
        <xs:element name="portInstance" type="concreteTypeInstance"/>
        <xs:element name="resourceInstance" type="concreteTypeInstance"/>
        <xs:element name="member" type="member"/>
        <xs:element name="containmentInstance" type="containmentInstance"/>
        <xs:element name="communicationInstance" type="communicationInstance"/>
        <xs:element name="hostingInstance" type="hostingInstance"/>
        <xs:element name="delegationInstance" type="delegationInstance"/>
        <xs:element name="referenceInstance" type="referenceInstance"/>
        <xs:element name="placementInstance" type="placementInstance"/>
    </xs:choice>
</xs:group>
```

[0836]

[0837] 3.14.7 인스턴스 참조

[0838] 3.14.7.1 인스턴스 Ref

[0839] 인스턴스 ref는 인스턴스에 대한 간단한 참조이다. 참조가 변경 요구의 컨텍스트내에서 작성되지 않으면 isCurrent 인스턴스를 디폴트로 하고, 인스턴스는 변경 요구에 의해 영향을 받는다.

```
<xs:complexType name="instanceRef">
    <xs:attribute name="instanceID" type="instanceID" use="required"/>
</xs:complexType>
```

[0840]

[0841] 3.14.7.2 인스턴스 버전 Ref

[0842] 인스턴스 버전 ref는 인스턴스의 특정 버전을 식별한다.

```
<xs:complexType name="instanceVersionRef">
    <xs:attribute name="instanceID" type="instanceID" use="required"/>
    <xs:attribute name="version" type="xs:int" use="required"/>
</xs:complexType>
```

[0843]

[0844] 3.15 배치 유닛 구조

[0845] 요구사항

[0846] • SDM 타입의 세트를 설치하기 위하여 모든 비트 요구를 포함한다.

[0847] • 사인 또는 버전될 수 있다.

[0848] • 용이하게 구성/패키지/이동된다.

[0849] • 참조 또는 포함에 의해 다른 SDU를 조회할 수 있다.

[0850] • SDM 타입 정의의 배치 섹션은 SDU 내의 파일을 직접 참조한다.

[0851] **3.16 로컬리제이션**

[0852] SDM 모델의 어느 부분이 로컬리제이션을 지원하고 시스템의 설계 및 배치를 통해 로컬리제이션을 어떻게 지원하는지를 결정할 필요가 있다.

[0853] 제1 접근:

[0854] 우리는 개별 타입 및 타입까지 로컬리제이션을 관리하도록 한다. 로컬리제이션은 제한을 통해 암시된다. 로컬리제이션은 제1 타입 시민(citizen)이다. 이것은 다음을 의미한다.

[0855] a) SDU는 특정 버전의 타입의 구현을 포함할 수 있다: 특정 버전의 일구현예가 있다. 이것은 로컬리제이션에 기초하여 다른 구현이 있을 수 없음을 의미한다. 각각의 구현은 장소의 범위를 지원해야 하거나 구현은 다른 타입이어야 한다(이 목적을 위하여 버저닝을 사용하는 것은 위반행위이다)

[0856] b) 로컬리제이션은 mixins로서 리소스를 사용하여 특정 버전을 지원하거나 다른 버전을 지원하는 구현을 식별하는 타입의 세트를 사용함으로써 달성된다.

[0857] c) 클라이언트는 서버의 로컬라이즈된 버전을 차별화/요구할 수 없다.

[0858] 제2 접근:

[0859] 로컬리제이션은 이름과 버전에 따른 식별의 제1 타입 시민(citizen)이다. 이것은 로컬리제이션이 참조가 타입에 작성되는 장소를 고려해야 하는 것을 의미한다.

[0860] a) 클라이언트는 포함, 호스팅 또는 통신 관계 중의 임의의 것에 대한 서버의 로컬라이즈된 버전을 차별화할 수 있다.

[0861] b) 배치 엔진은 로컬리제이션을 알아야 하고 오퍼레이터가 타입의 로컬라이즈된 버전들 사이에서 선택하도록 한다.

[0862] c) SDU는 이름, 버전 및 장소(들)에 의해 식별되거나 SDU는 그들의 장소에 기초하여 다른 다수의 구현을 포함할 수 있다(첫번째는 로컬라이즈되지 않은 코드가 별개의 sdu에 배치되어야 하므로 SDU의 더 미세한 패키징을 암시하고, 두번째는 동일한 이름을 갖는 다수의 sdu를 가질 수 있다는 것을 의미한다)

[0863] 제2 접근은 장소가 제한으로서 넓게 사용되면 설계/ui 관점으로부터 매우 복잡한 잠재력을 갖는다. 예를 들어, 엔드포인트가 로컬라이즈되거나 호스트가 그들의 게스트를 로컬라이즈하면, 접속/배치를 탐색하는 것은 더 복잡하게 한다. 제2 접근은 제안된 메카니즘과 같이 제1 접근으로부터 b)로 사용되면, 복잡성은 더 쉽게 관리하지만 누군가가 로컬라이즈된 리소스를 식별, 패키징 및 이동해야 한다.

[0864] **3.17 버저닝 및 변경 관리**

[0865] **3.17.1 일반적인 코멘트**

[0866] • 우리는 적절하게 시스템을 버저닝할 수 있기를 원한다 - 즉, 인스턴스 식별을 변경하지 않고 qfe를 sql에 적용한다. 이것은 인스턴스의 타입을 변경하는 것을 암시한다.

[0867] • 우리는 버저닝 정책이 허용된 버전 변경을 허용하기를 원한다 - 예를 들어, 시스템 타입 설계자는 시스템의 멤버에 대한 버저닝 정책이 얼마나 엄격한지를 선택할 수 있거나 오퍼레이터가 보안 이유로 멤버의 버전을 일방적으로 업그레이드하도록 선택할 수 있다.

[0868] • 우리는 버저닝 변경의 전파를 제한하기를 워한다 - 예를 들어, 우리가 멤버의 타입을 변경하면, 우리는 시스

템의 타입을 변경하는 것을 원하지 않고, 따라서 타입 변경을 루트에 전파한다.

[0869] • 브레이킹 변경은 버전 번호의 첫번째 2 부분의 변경에 의해 표시될 것이며, 비-브레이킹 변경은 버전 번호의 두번째 2 부분의 변경에 의해 표시될 것이다.

[0870] 컴퓨터 환경의 예

[0871] 도 21은 여기에 기재된 기술을 구현하는 데 사용될 수 있는 일반적인 컴퓨터 환경(600)을 나타낸다. 컴퓨터 환경(600)은 컴퓨팅 환경의 일례일 뿐, 컴퓨터와 네트워크 아키텍처의 기능 또는 사용의 범위로 제한하는 것이 아니다. 컴퓨터 환경(600)은 예시적인 환경(600)에 예시된 컴포넌트 중의 임의의 하나 또는 조합에 관련된 임의의 종속성 또는 요구사항으로서 해석되어서는 안된다.

[0872] 컴퓨터 환경(600)은 컴퓨터(602)의 형태의 범용 컴퓨터 장치를 포함한다. 컴퓨터(602)는 예를 들어 도 1의 컴퓨팅 장치(102)일 수 있거나, 개발 시스템(202)을 구현할 수 있거나, 도 2의 컨트롤러(206)일 수 있거나, 도 2의 타겟 장치(212)이거나 도 5의 컨트롤러(520) 또는 타겟(522)일 수 있다. 컴퓨터(602)의 컴포넌트는 하나 이상의 프로세서 또는 프로세싱 유닛(604), 시스템 메모리(606), 및 프로세서(604)를 포함하는 다양한 시스템 컴포넌트를 시스템 메모리(606)에 결합하는 시스템 버스(608)를 포함하지만, 이에 한정되는 것은 아니다.

[0873] 시스템 버스(608)는 메모리 버스 또는 메모리 컨트롤러, 주변 버스, 가속 그래픽 포트, 및 다양한 버스 아키텍처 중의 임의의 것을 사용한 프로세서 또는 로컬 버스를 포함하는 몇가지 유형의 버스 구조 중 하나 이상을 나타낸다. 예로서, 이러한 아키텍처는 산업 표준 아키텍처(ISA) 버스, 마이크로 채널 아키텍처(MCA) 버스, 인핸스드 ISA(EISA; enhanced ISA) 버스, 비디오 일렉트로닉스 표준 어소시에이션(VESA) 로컬 버스, 및 메자닌 버스(Mezzanine bus)로 공지된 주변 컴포넌트 상호접속(PCI) 버스를 포함할 수 있다.

[0874] 컴퓨터(602)는 일반적으로 다양한 컴퓨터 판독가능 매체를 포함한다. 이러한 매체는 컴퓨터(602)에 의해 액세스 가능하고 휘발성 및 비휘발성 매체 및 분리형 및 비분리형 매체를 포함하는 임의의 이용가능한 매체일 수 있다.

[0875] 시스템 메모리(606)는 랜덤 액세스 메모리(RAM; 610)와 같은 휘발성 메모리, 및/또는 판독 전용 메모리(ROM; 612)와 같은 비휘발성 메모리 형태의 컴퓨터 판독가능 매체를 포함한다. 시동 등과 같이 컴퓨터(602) 내의 소자들간에 정보를 전송하는 것을 돕는 기본 루틴을 포함하는 기본 입출력 시스템(BIOS; 614)이 ROM(612)에 저장된다. RAM(610)은 통상 프로세싱 유닛(604)에 즉시 액세스 가능하고 및/또는 프로세싱 유닛에 의해 동작하는 데이터 및/또는 프로그램 모듈을 포함한다.

[0876] 컴퓨터(602)는 또한 다른 분리형/비분리형, 휘발성/비휘발성 컴퓨터 기억 매체를 포함할 수 있다. 예로서, 도 21은 비분리형 비휘발성 자기 매체(도시하지 않음)으로부터 판독되고 그 자기 매체에 기록하는 하드 디스크 드라이브(616), 분리형 비휘발성 자기 디스크(620)(예를 들어, "플로피 디스크")로부터 판독하고 그 자기 디스크에 기록하는 자기 디스크 드라이브(618), 및 CD-ROM, DVD-ROM 또는 다른 광학 매체와 같은 분리형 비휘발성 광학 디스크(624)로부터 판독하고 및/또는 기록하는 광학 디스크 드라이브(622)를 도시한다. 하드 디스크 드라이브(616), 자기 디스크 드라이브(618), 및 광학 디스크 드라이브(622)는 각각 하나 이상의 데이터 매체 인터페이스(626)에 의해 시스템 버스(608)에 접속된다. 다른 방법으로, 하드 디스크 드라이브(616), 자기 디스크 드라이브(618), 및 광학 디스크 드라이브(622)는 하나 이상의 인터페이스(도시하지 않음)에 의해 시스템 버스(608)에 접속될 수 있다.

[0877] 디스크 드라이브 및 그 관련 컴퓨터 판독가능 매체는 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 및 컴퓨터(602)를 위한 다른 데이터의 비휘발성 기억장치를 제공한다. 상기 예는 하드 디스크(616), 분리형 자기 디스크(620) 및 분리형 광학 디스크(624)를 예시하지만, 자기 카세트 또는 다른 자기 기억 장치, 플래시 메모리 카드, CD-ROM, DVD(digital versatile disk) 또는 다른 광학 기억장치, 랜덤 액세스 메모리(RAM), 판독 전용 메모리(ROM), 전기적 소거가능 프로그램가능 판독 전용 메모리(EEPROM) 등의 컴퓨터에 의해 액세스가능한 데이터를 저장할 수 있는 다른 유형의 컴퓨터 판독 가능 매체가 예시적인 컴퓨팅 시스템 및 환경을 구현하기 위하여 이용될 수 있음은 자명하다.

[0878] 예로서 오퍼레이팅 시스템(626), 하나 이상의 애플리케이션 프로그램(628), 다른 프로그램 모듈(630) 및 프로그램 데이터(632)를 포함하는 임의의 수의 프로그램 모듈은 하드 디스크(616), 자기 디스크(620), 광학 디스크(624), ROM(612) 및/또는 RAM(610)에 저장될 수 있다. 이러한 오퍼레이팅 시스템(626), 하나 이상의 애플리케이션 프로그램(628), 다른 프로그램 모듈(630) 및 프로그램 데이터(632)의 각각(또는 그들의 일부 조합)은 분산

형 파일 시스템을 지원하는 상주 컴포넌트의 모두 또는 부분을 구현할 수 있다.

[0879] 사용자는 키보드(634) 및 포인팅 장치(636; 예를 들어 "마우스") 등의 입력 장치를 통해 컴퓨터(602)에 코맨드 및 정보를 입력할 수 있다. 다른 입력 장치(638)(특별히 도시하지 않음)는 마이크로폰, 조이스틱, 게임 패드, 위성 안테나, 시리얼 포트, 스캐너 등을 포함할 수 있다. 이들 및 다른 입력 장치는 시스템 버스(608)에 결합된 입출력 인터페이스(640)를 통해 프로세싱 유닛(604)에 접속되지만, 병렬 포트, 게임 포트 또는 유니버설 시리얼 버스(USB) 등의 다른 인터페이스 및 버스 구조에 의해 접속될 수 있다.

[0880] 모니터(642) 또는 다른 유형의 디스플레이 장치는 비디오 어댑터(644) 등의 인터페이스를 통해 시스템 버스(608)에 접속될 수 있다. 모니터(642)에 더하여, 다른 출력 주변 장치는 입출력 인터페이스(640)를 통해 컴퓨터(602)에 접속될 수 있는 프린터 및 스피커(도시하지 않음) 등의 컴포넌트를 포함할 수 있다.

[0881] 컴퓨터(620)는 원격 컴퓨팅 장치(648) 등의 하나 이상의 원격 컴퓨터로의 논리적 접속을 사용하는 네트워크된 환경에서 동작할 수 있다. 예로서, 원격 컴퓨팅 장치(648)는 퍼스널 컴퓨터, 포터블 컴퓨터, 서버, 라우터, 네트워크 컴퓨터, 피어 장치 또는 다른 공통 네트워크 노드일 수 있다. 원격 컴퓨팅 장치(648)는 컴퓨터(602)에 관련된 여기에 기재된 특징과 요소의 다수 또는 모두를 포함할 수 있는 포터블 컴퓨터로서 도시된다.

[0882] 컴퓨터(602) 및 원격 컴퓨터(648)간의 논리적 접속은 근거리 통신망(LAN; 650) 및 일반적인 원거리 통신망(WAN; 652)으로서 도시된다. 이러한 네트워킹 환경은 사무실, 기업 광역 컴퓨터 네트워크, 인트라넷 및 인터넷에서 일반적인 것이다.

[0883] LAN 네트워킹 환경에서 구현될 때, 컴퓨터(602)는 네트워크 인터페이스 또는 어댑터(654)를 통해 근거리 통신망(650)에 접속된다. WAN 네트워킹 환경에서 구현될 때, 컴퓨터(602)는 통상 원거리 통신망(652)를 통해 통신을 구축하는 모뎀(656) 또는 다른 수단을 포함한다. 컴퓨터(602)에 내부적 또는 외부적일 수 있는 모뎀(656)은 입출력 인터페이스(640) 또는 다른 적절한 메커니즘을 통해 시스템 버스(608)에 접속될 수 있다. 도시된 접속은 예시적인 것이며 컴퓨터(602, 648) 간의 통신 구축을 위한 다른 수단이 사용될 수 있다.

[0884] 컴퓨팅 환경(600)에 도시된 네트워킹된 환경에서, 컴퓨터(602)와 관련된 프로그램 모듈 또는 그 부분이 원격 메모리 기억 장치에 저장될 수 있다. 예로서, 원격 애플리케이션 프로그램(658)은 원격 컴퓨터(648)의 메모리 장치 상에 상주한다. 이러한 프로그램 및 컴포넌트는 많은 경우 컴퓨팅 장치(602)의 다른 기억 컴포넌트에 상주하고 컴퓨터의 데이터 프로세서(들)에 의해 실행되는 것으로 인식되지만, 설명을 목적으로, 애플리케이션 프로그램 및 오퍼레이팅 시스템과 같은 다른 실행가능 프로그램 컴포넌트는 별개의 블록으로 도시되었다.

[0885] 다양한 모듈 및 기술은 하나 이상의 컴퓨터 또는 다른 장치에 의해 실행되는 프로그램 모듈 등의 컴퓨터 실행가능 명령의 일반적인 컨텍스트에서 기재될 수 있다. 일반적으로, 프로그램 모듈은 특정 태스크를 수행하거나 특정 추상 데이터 타입을 구현하는 루틴, 프로그램, 객체, 컴포넌트, 데이터 구조 등을 포함한다. 통상, 프로그램 모듈의 기능은 많은 실시예에서 바람직하게 결합 또는 분산될 수 있다.

[0886] 이들 모듈 및 기술의 구현은 임의의 형태의 컴퓨터 판독가능 매체 상에 저장되거나 컴퓨터 판독가능 매체를 통해 전송될 수 있다. 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 이용가능한 매체일 수 있다. 예로서, 한정되지는 않지만, 컴퓨터 판독 가능 매체는 "컴퓨터 기억 매체" 및 "통신 매체"를 포함할 수 있다.

[0887] "컴퓨터 기억 매체"는 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 또는 다른 데이터 등의 정보의 기억을 위한 임의의 방법 또는 기술에 의해 구현되는 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 포함한다. 컴퓨터 기억 매체는 RAM, ROM, EEPROM, 플래시 메모리 또는 다른 메모리 기술, CD-ROM, DVD 또는 다른 광학 기억장치, 자기 카세트, 자기 테이프, 자기 디스크 기억장치 또는 다른 자기 기억 장치 또는 원하는 정보를 저장하는 데 사용될 수 있고 컴퓨터에 의해 액세스될 수 있는 임의의 다른 매체를 포함하며, 이로는 한정되지 않는다.

[0888] "통신 매체"는 통상 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 또는 반송파 또는 다른 전송 메카니즘 등의 변조 데이터 신호의 다른 데이터를 구현한다. 통신 매체는 또한 임의의 정보 전달 매체를 포함한다. 용어 "변조 데이터 신호"는 정보를 신호로 인코딩하는 방식으로 변경되거나 설정된 그 특성들 중의 하나 이상을 갖는 신호를 의미한다. 예로서, 한정되지는 않지만, 통신 매체는 유선 네트워크, 직접 유선 접속 등의 유선 매체 및 음향, RF, 적외선 및 다른 무선 매체와 같은 무선 매체를 포함한다. 상술한 것들의 임의의 조합은 또한 컴퓨터 판독가능 매체의 범위내에 포함된다.

[0889] 대안으로, 프레임워크의 부분은 하드웨어 또는 하드웨어, 소프트웨어, 및/또는 펌웨어의 조합으로 구현될 수 있다. 예를 들어, 하나 이상의 응용 주문형 집적 회로(ASIC) 또는 프로그램가능 논리 장치(PLD)가 프레임워크의 하나 이상의 부분을 구현하도록 설계되거나 프로그램될 수 있다.

[0890] 결론

[0891] 본 발명은 구조적 특징 및/또는 방법론적 액션에 특정된 언어로 기재되었지만, 첨부된 청구항에 정의된 본 발명은 기재된 특정 특징 또는 액션으로 반드시 제한되는 것은 아니다. 오히려, 특정 특징 및 액션은 청구한 발명을 구현하는 예시적인 형태로서 개시된다.

발명의 효과

[0892] 상술한 바와 같이, 본 발명에 따르면, 물리적 컴퓨팅 시스템 상에 분산형 애플리케이션을 설계하고 배치하는 향상된 기술이 제공될 수 있다.

도면의 간단한 설명

[0001] 도 1은 네트워크 세팅의 일례를 나타내는 도면.

[0002] 도 2는 SDM 정의 모델을 사용하는 아키텍처의 일례를 나타내는 블록도.

[0003] 도 3은 계층화된 세팅(layered setting)의 일례를 나타내는 도면.

[0004] 도 4는 시스템의 전체 라이프사이클에 걸친 시스템 정의 모델(SDM)을 사용하는 프로세스의 일례를 나타내는 플로우챠트.

[0005] 도 5는 SDM 런타임을 사용하는 아키텍처의 일례를 나타내는 도면.

[0006] 도 6은 SDM 문서의 일례를 나타내는 도면.

[0007] 도 7은 기본 정의와 멤버를 나타내는 도면.

[0008] 도 8은 멤버의 일례를 나타내는 도면.

[0009] 도 9는 세팅 값과 값 리스트의 예를 나타내는 도면.

[0010] 도 10은 소정의 실시예에 따른 SDM 애플리케이션의 라이프사이클의 일례를 나타내는 도면.

[0011] 도 11은 웹 서버 호스트에 웹 애플리케이션을 맵핑하는 일례를 나타내는 도면.

[0012] 도 12는 내장 데이터타입 계층의 일례를 나타내는 도면.

[0013] 도 13은 추상 객체 정의의 암시적 확장의 일례를 나타내는 도면.

[0014] 도 14는 추상 관계의 암시적 확장의 일례를 나타내는 도면.

[0015] 도 15는 변경 요구의 일례를 나타내는 도면.

[0016] 도 16은 새로운 정의를 런타임으로 로딩하는 프로세스의 일례를 나타내는 도면.

[0017] 도 17은 변경 요구를 수행하는 일례를 나타내는 도면.

[0018] 도 18은 접속된 멤버의 예를 나타내는 도면.

[0019] 도 19는 접속과 관련된 구조의 예를 나타내는 도면.

[0020] 도 20은 인스턴스 공간의 개요를 제공하는 UML 다이어그램의 일례를 나타내는 도면.

[0021] 도 21은 여기에 기재된 기술을 구현하는 데 사용될 수 있는 일반적인 컴퓨터 환경을 나타내는 도면.

[0022] <도면의 주요부분에 대한 부호의 설명>

[0023] 200: 아키텍처

[0024] 202: 개발 시스템

[0025] 204: SDM 문서

[0026] 206: 컨트롤러

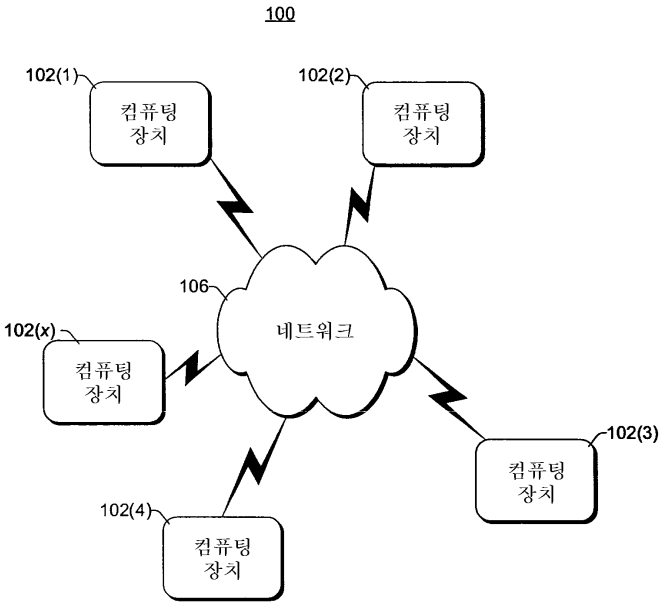
[0027] 208: 배치 모듈

[0028] 210: 관리 모듈

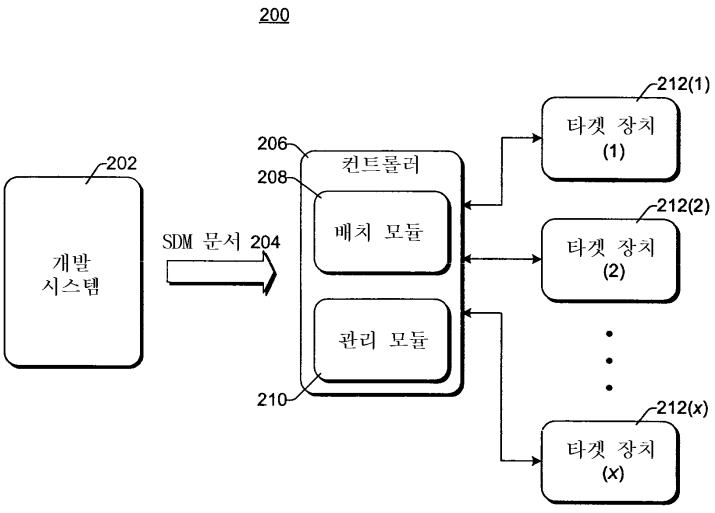
[0029] 212: 타겟 장치

도면

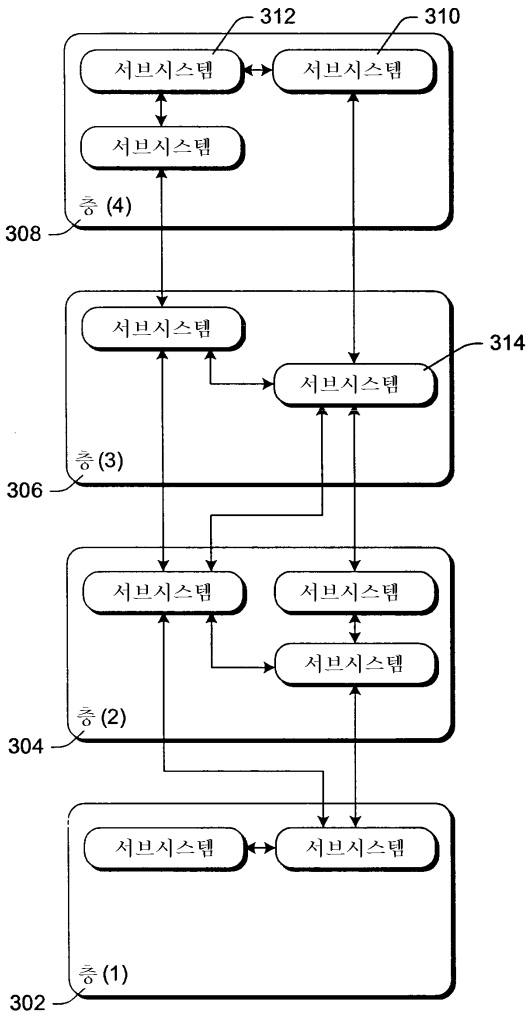
도면1



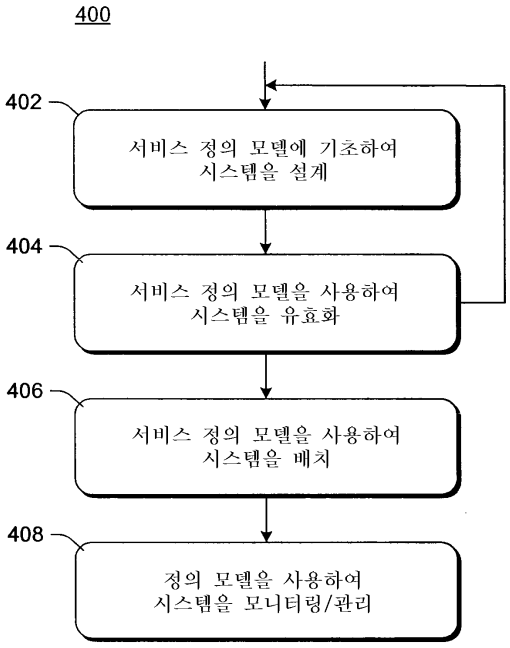
도면2



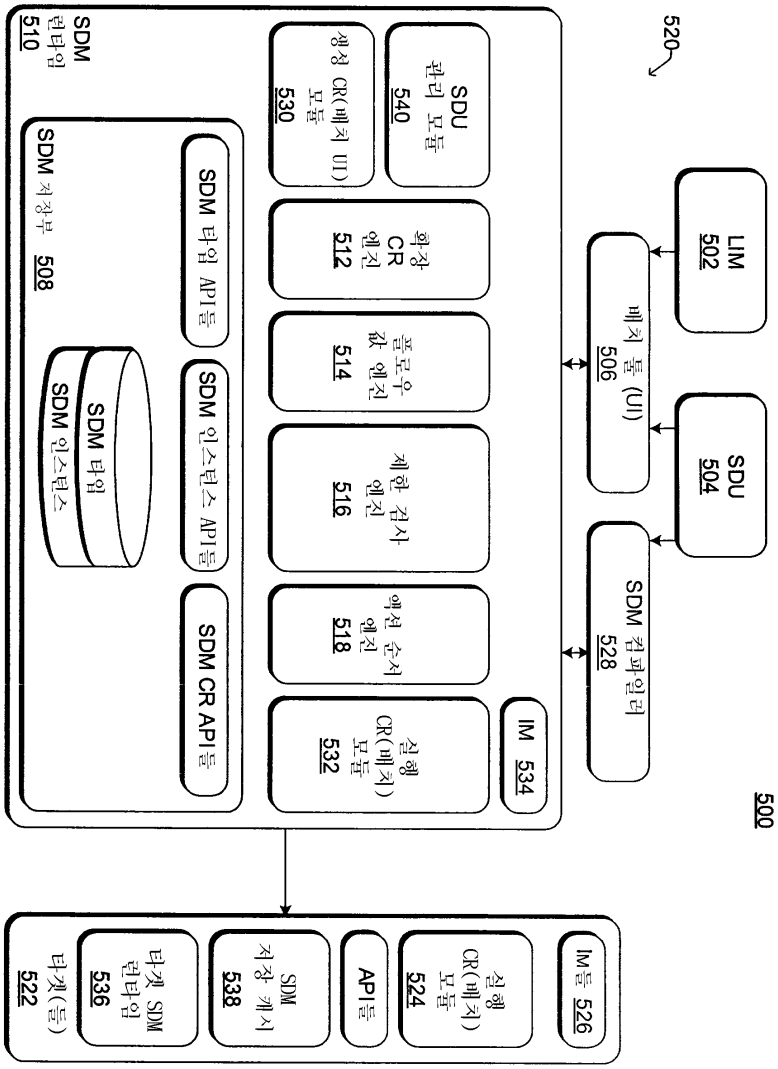
도면3


312
310
서브시스템
서브시스템
서브시스템
층 (4)
308
서브시스템
314
서브시스템
층 (3)
306
서브시스템
서브시스템
서브시스템
층 (2)
304
서브시스템
서브시스템
층 (1)
302

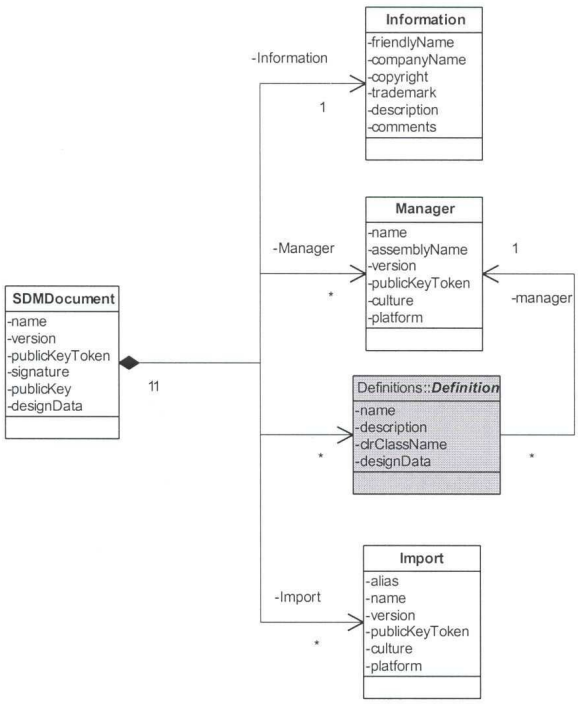
도면4


400
402
서비스 정의 모델에 기초하여
시스템을 설계
404
서비스 정의 모델을 사용하여
시스템을 유효화
406
서비스 정의 모델을 사용하여
시스템을 배치
408
정의 모델을 사용하여
시스템을 모니터링/관리

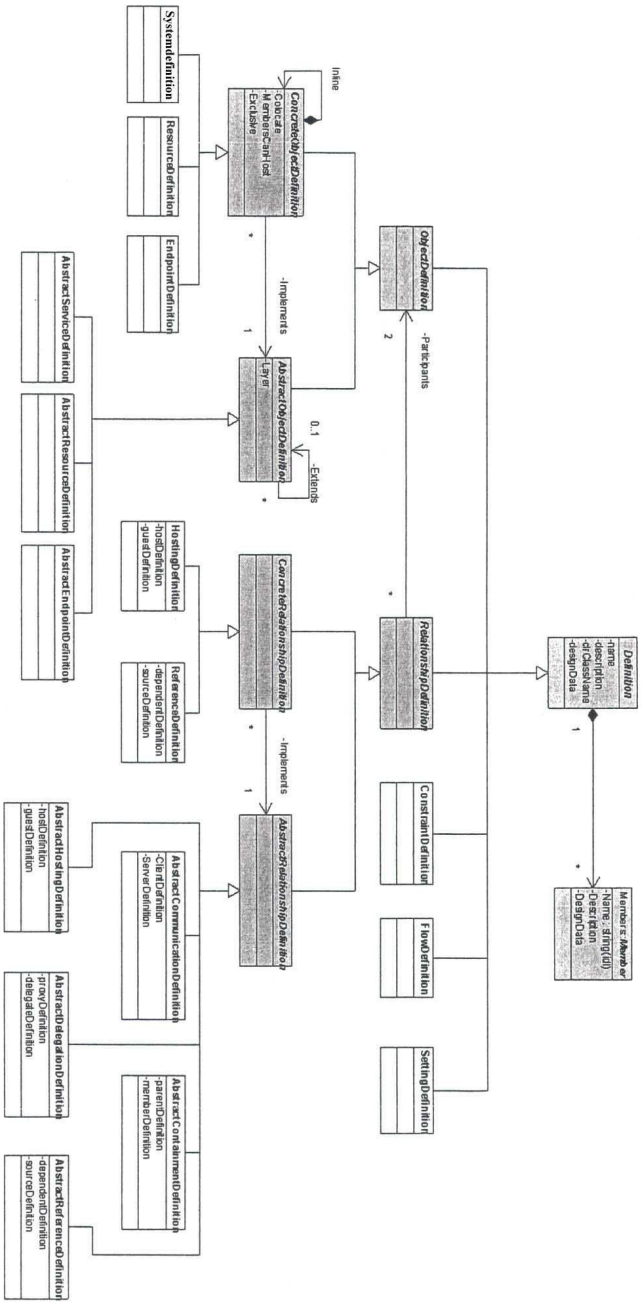
도면5


500
520
LIM 502
SDU 504
배치 툴 (UI) 506
SDM 컴파일러 528
SDU 관리 모듈 540
생성 CR(배치 UI) 모듈 530
확장 CR 엔진 512
플로우 값 엔진 514
제한 검사 엔진 516
엑션 순서 엔진 518
실행 CR(배치) 모듈 532
IM 534
SDM 런타임 510
SDM 저장부 508
SDM 타입
SDM 인스턴스
SDM 타입 API들
SDM 인스턴스 API들
SDM CR API들
IM들 526
실행 CR(배치) 모듈 524
API들
SDM 저장 캐시 538
타겟 SDM 런타임 536
타겟(들) 522

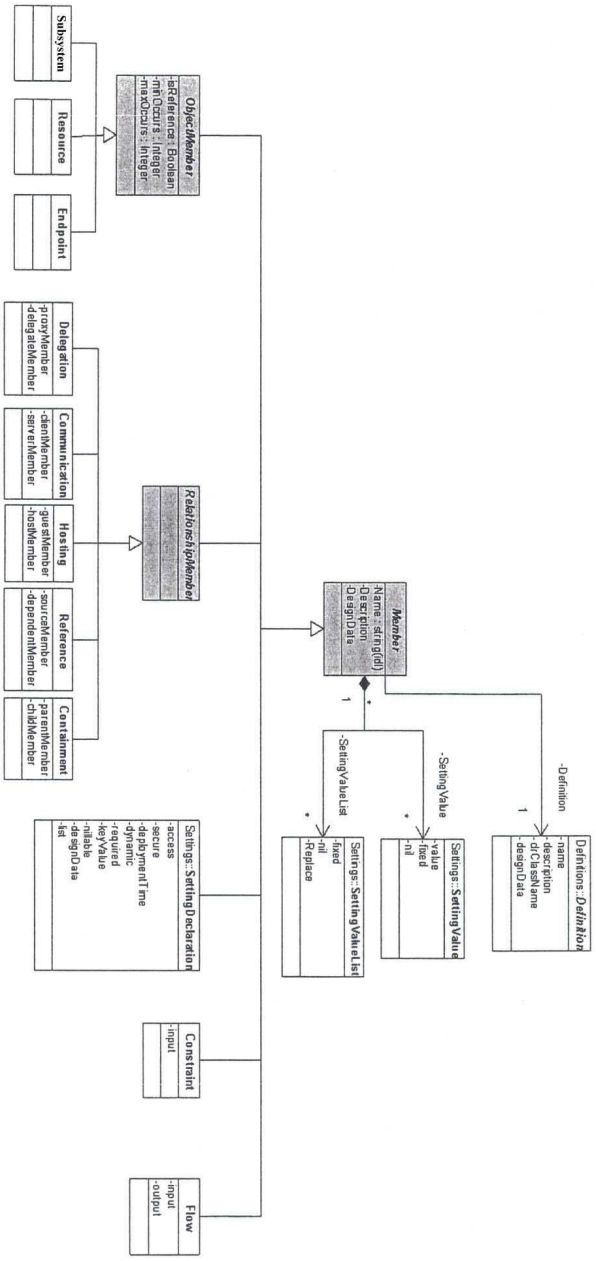
도면6


Information
-friendlyName
-companyName
-copyright
-trademark
-description
-comments
-Information
1
Manager
-name
-assemblyName
-version
-publicKeyToken
-culture
-platform
-Manager
*
1
-manager
SDMDocument
-name
-version
-publicKeyToken
-signature
-publicKey
-designData
11
Definitions::Definition
-name
-description
-drClassName
-designData
*
*
Import
-alias
-name
-version
-publicKeyToken
-culture
-platform
-Import
*

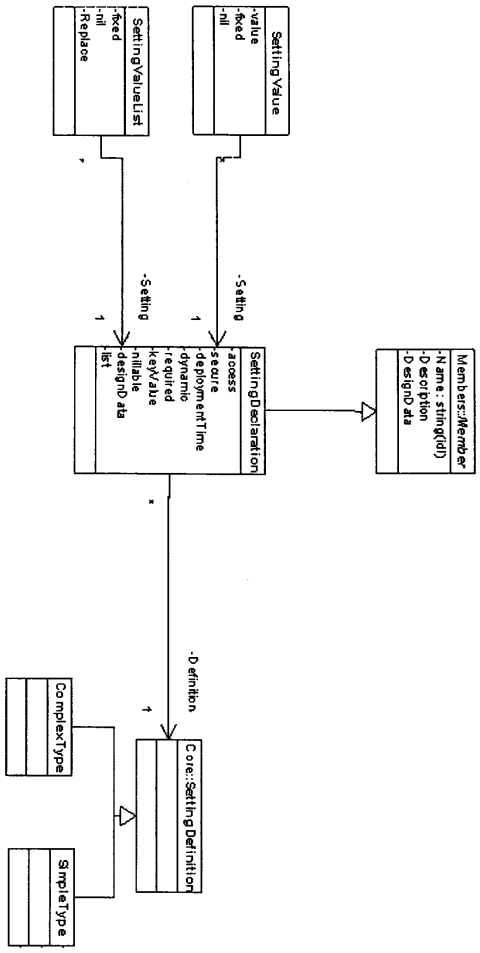
도면7


Definition
-name
-description
-dtCreationTime
-designData
Members:Member
-Name:string(id)
-Description
-DesignData
1
*
ObjectDefinition
RelationshipDefinition
ConstraintDefinition
FlowDefinition
SettingDefinition
-Participants
2
*
Inline
ConcreteObjectDefinition
-Category
-IsInstanceCapable
-Extends
SystemDefinition
ResourceDefinition
EndpointDefinition
-Implements
1
*
AbstractObjectDefinition
-Layer
0..1
-Extends
AbstractServiceDefinition
AbstractResourceDefinition
AbstractEndpointDefinition
ConcreteRelationshipDefinition
HostingDefinition
-hostDefinition
-guestDefinition
ReferenceDefinition
-dependentDefinition
-sourceDefinition
-Implements
AbstractRelationshipDefinition
AbstractHostingDefinition
-hostDefinition
-guestDefinition
AbstractCommunicationDefinition
-ClientDefinition
-ServerDefinition
AbstractDelegationDefinition
-proxyDefinition
-delegateDefinition
AbstractContainmentDefinition
-parentDefinition
-memberDefinition
AbstractReferenceDefinition
-dependentDefinition
-sourceDefinition

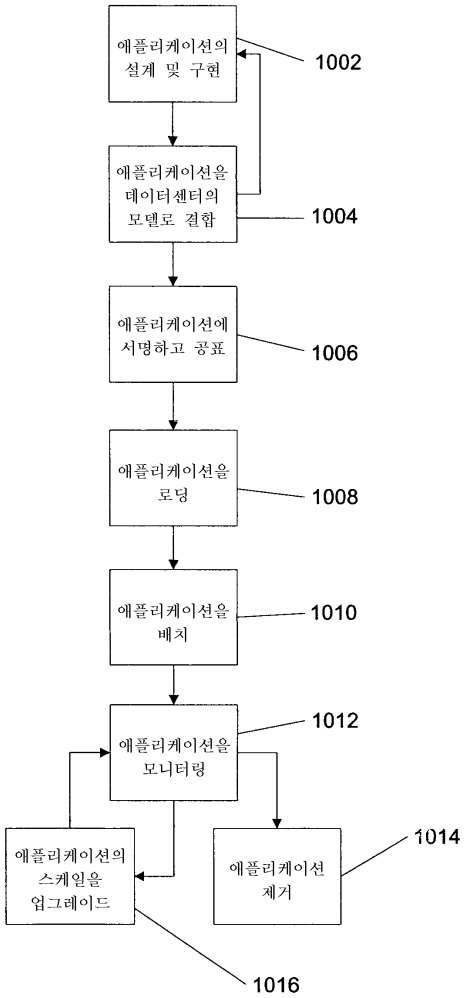
도면8


Subsystem
Resource
Endpoint
ObjectMember
-isReference : Boolean
-minOccurs : Integer
-maxOccurs : Integer
Delegation
-proxyMember
-delegateMember
Communication
-clientMember
-serverMember
Hosting
-guestMember
-hostMember
Reference
-sourceMember
-dependentMember
Containment
-parentMember
-childMember
RelationshipMember
Member
-Name : string{id}
-Description
-DesignData
1
*
-SettingValueList
-SettingValue
-Definition
Settings::SettingValueList
-fixed
-nil
-Replace
Settings::SettingValue
-value
-fixed
-nil
Definitions::Definition
-name
-description
-dfClassName
-designData
Settings::SettingDeclaration
-access
-secure
-deploymentTime
-dynamic
-required
-keyValue
-nillable
-designData
-list
Constraint
-input
Flow
-input
-output

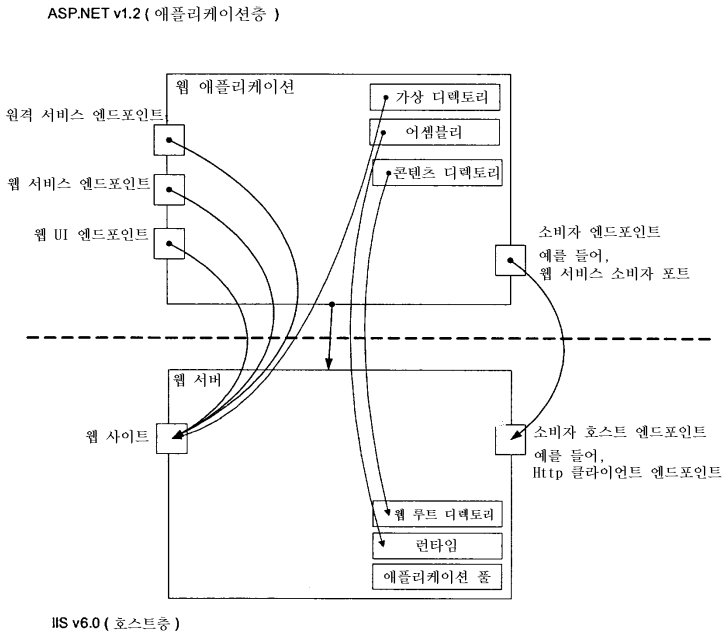
도면9


SettingValueList
-fixed
-nil
-Replace
SettingValue
-value
-fixed
-nil
-Setting
-Setting
SettingDeclaration
-access
-secure
-deploymentTime
-dynamic
-required
-keyValue
-nillable
-designData
-list
Members::Member
-Name : string(id)
-Description
-designData
-Definition
Core::SettingDefinition
ComplexType
SimpleType

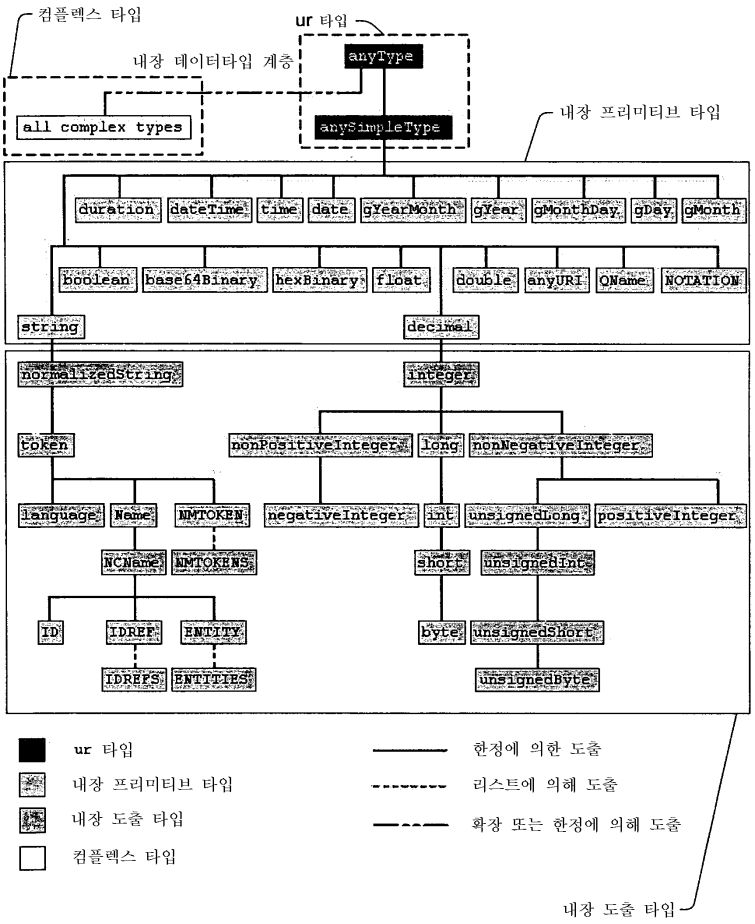
도면10


애플리케이션의 설계 및 구현
1002
애플리케이션을 데이터센터의 모델로 결합
1004
애플리케이션에 서명하고 공표
1006
애플리케이션을 로딩
1008
애플리케이션을 배치
1010
애플리케이션을 모니터링
1012
애플리케이션의 스케일을 업그레이드
애플리케이션 제거
1014
1016

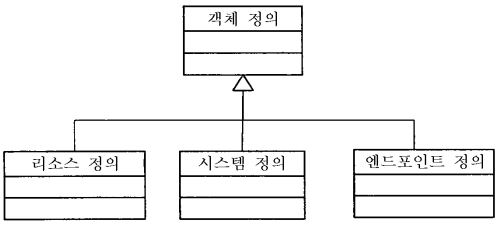
도면11


ASP.NET v1.2 (애플리케이션층)
웹 애플리케이션
원격 서비스 엔드포인트
웹 서비스 엔드포인트
웹 UI 엔드포인트
가상 디렉토리
어셈블리
콘텐츠 디렉토리
소비자 엔드포인트
예를 들어,
웹 서비스 소비자 포트
웹 서버
웹 사이트
소비자 호스트 엔드포인트
예를 들어,
Http 클라이언트 엔드포인트
웹 루트 디렉토리
런타임
애플리케이션 풀
IIS v6.0 (호스트층)

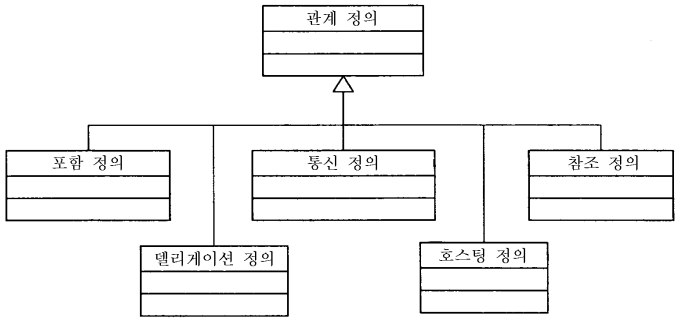
도면12


컴플렉스 타입
ur 타입
내장 데이터타입 계층
anyType
all complex types
anySimpleType
내장 프리미티브 타입
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur 타입
내장 프리미티브 타입
내장 도출 타입
컴플렉스 타입
한정에 의한 도출
리스트에 의해 도출
확장 또는 한정에 의해 도출
내장 도출 타입

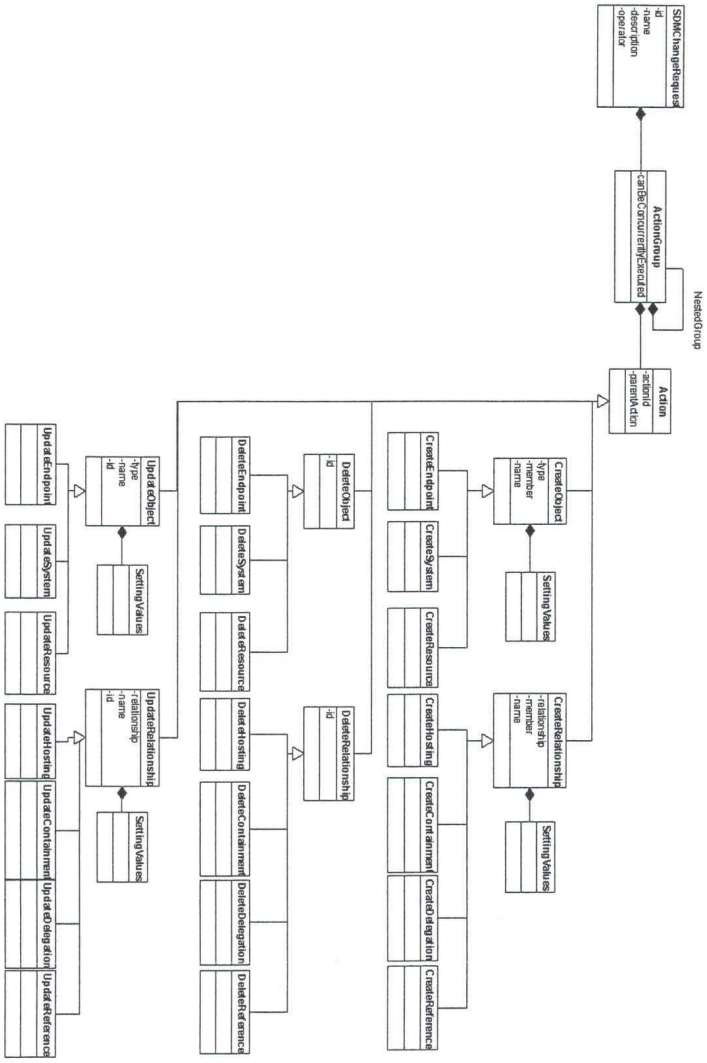
도면13


객체 정의
리소스 정의
시스템 정의
엔드포인트 정의

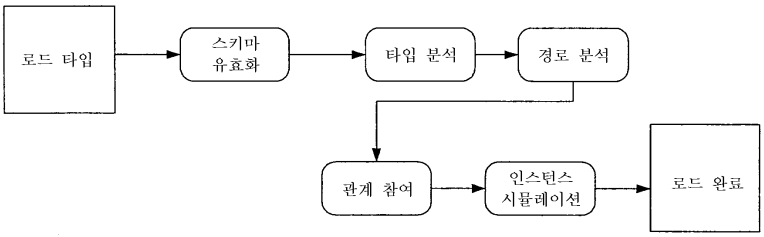
도면14


관계 정의
포함 정의
통신 정의
참조 정의
델리게이션 정의
호스팅 정의

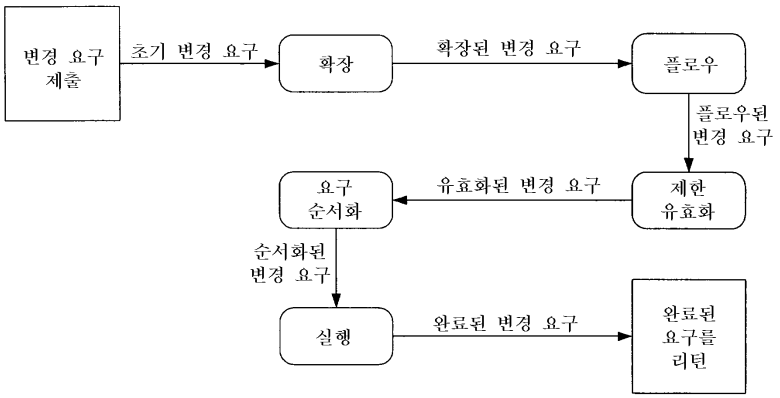
도면15


SDMCchangeRequest
-id
-name
-description
-operator
ActionGroup
-canBeConcurrentlyExecuted
NestedGroup
Action
-actionId
-parentAction
CreateObject
-type
-member
-name
SettingValues
CreateEndpoint
CreateSystem
CreateResource
CreateHosting
CreateContainment
CreateDelegation
CreateReference
CreateRelationship
-relationship
-member
-name
SettingValues
DeleteObject
-id
DeleteEndpoint
DeleteSystem
DeleteResource
DeleteHosting
DeleteContainment
DeleteDelegation
DeleteReference
DeleteRelationship
-id
UpdateObject
-type
-name
-id
SettingValues
UpdateEndpoint
UpdateSystem
UpdateResource
UpdateHosting
UpdateContainment
UpdateDelegation
UpdateReference
UpdateRelationship
-relationship
-name
-id
SettingValues

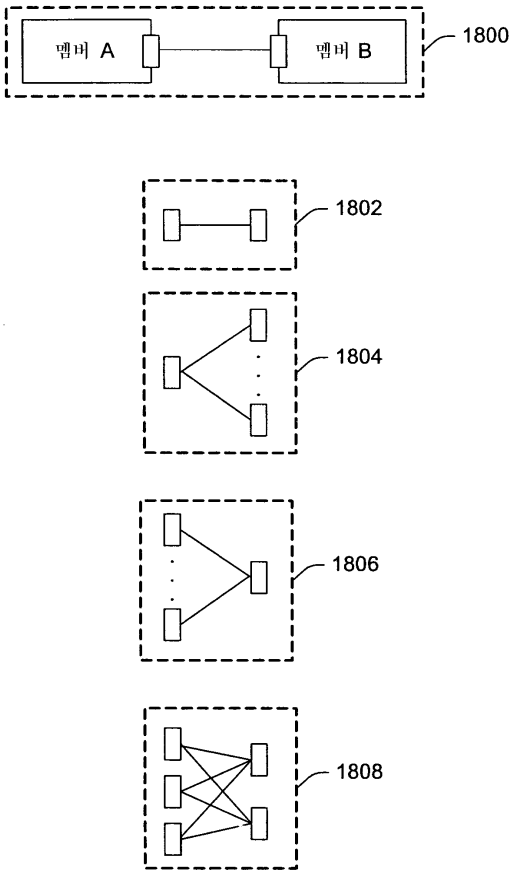
도면16


로드 타입
스키마 유효화
타입 분석
경로 분석
관계 참여
인스턴스 시뮬레이션
로드 완료

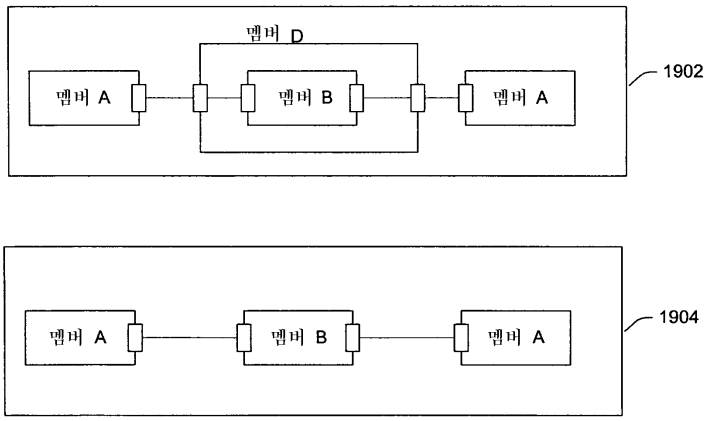
도면17


변경 요구 제출
초기 변경 요구
확장
확장된 변경 요구
플로우
플로우된 변경 요구
제한 유효화
유효화된 변경 요구
요구 순서화
순서화된 변경 요구
실행
완료된 변경 요구
완료된 요구를 리턴

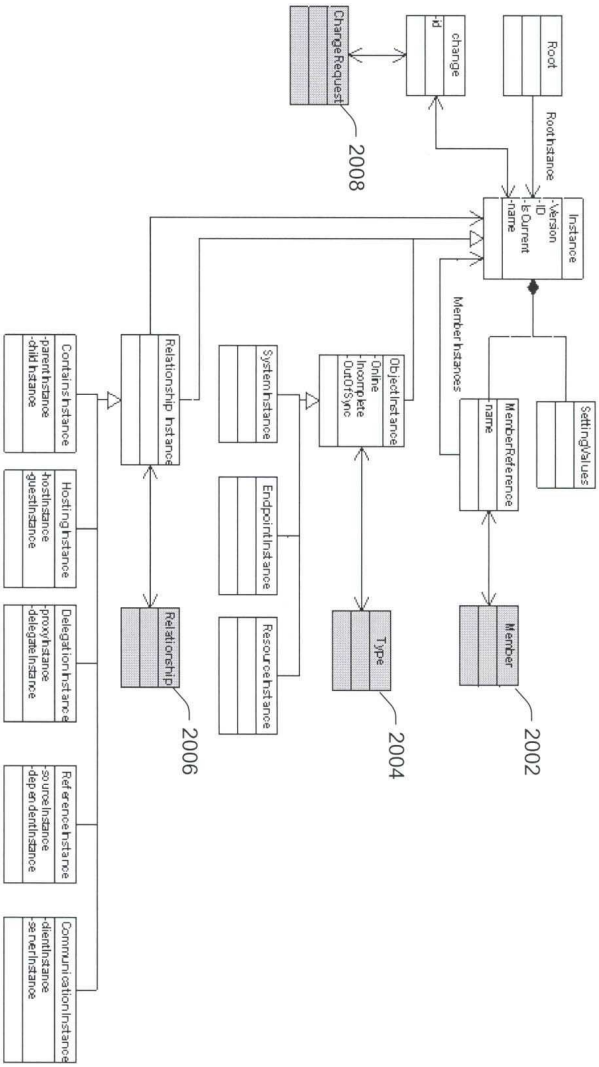
도면18


멤버 A
멤버 B
1800
1802
1804
1806
1808

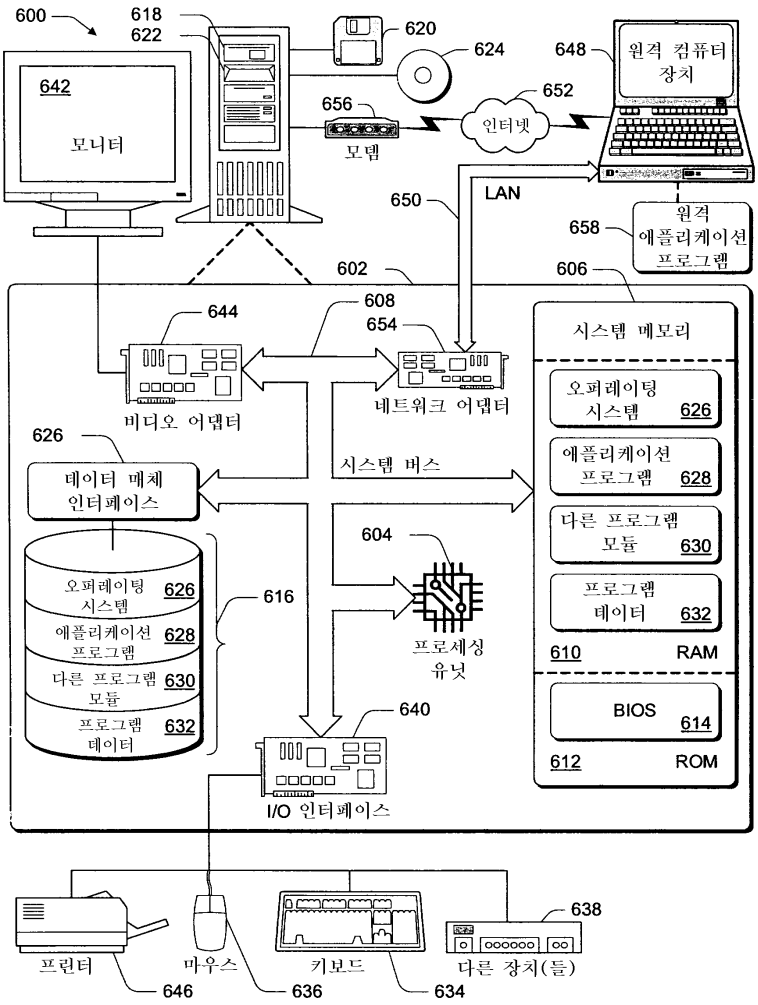
도면19


멤버 D
멤버 A
멤버 B
멤버 A
1902
멤버 A
멤버 B
멤버 A
1904

도면20


Root
RootInstance
Instance
-Version
-ID
-IsCurrent
-name
change
-id
ChangeRequest
2008
MemberInstances
MemberReference
-name
SettingValues
Member
2002
Type
2004
ObjectInstance
-Online
-Incomplete
-OutOfSync
SystemInstance
EndpointInstance
ResourceInstance
RelationshipInstance
Relationship
2006
ContainsInstance
-parentInstance
-childInstance
HostingInstance
-hostInstance
-guestInstance
DelegationInstance
-proxyInstance
-delegateInstance
ReferenceInstance
-sourceInstance
-dependentInstance
CommunicationInstance
-clientInstance
-serverInstance

도면21


600
618
622
620
624
648
원격 컴퓨터 장치
642
모니터
656
652
인터넷
모뎀
650
LAN
658
원격 애플리케이션 프로그램
602
606
644
608
654
시스템 메모리
비디오 어댑터
네트워크 어댑터
오퍼레이팅 시스템 626
626
데이터 매체 인터페이스
시스템 버스
애플리케이션 프로그램 628
다른 프로그램 모듈 630
604
오퍼레이팅 시스템 626
616
애플리케이션 프로그램 628
다른 프로그램 모듈 630
프로그램 데이터 632
프로그램 데이터 632
610 RAM
프로세싱 유닛
640
BIOS 614
612 ROM
I/O 인터페이스
638
프린터
마우스
키보드
다른 장치(들)
646
636
634