(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.[7]
G06F 17/30
G06F 17/00
G06F 15/16

(11) 공개번호 10-2005-0033077
(43) 공개일자 2005년04월08일

(21) 출원번호 10-2005-7003381
(22) 출원일자 2005년02월25일
번역문 제출일자 2005년02월25일
(86) 국제출원번호 PCT/US2003/026758
국제출원출원일자 2003년08월25일
(87) 국제공개번호 WO 2004/019238
국제공개일자 2004년03월04일

(30) 우선권주장
60/406,204 2002년08월26일 미국(US)
60/406,205 2002년08월26일 미국(US)
60/406,319 2002년08월26일 미국(US)
60/406,325 2002년08월26일 미국(US)
60/406,328 2002년08월26일 미국(US)
60/406,391 2002년08월26일 미국(US)
60/406,399 2002년08월26일 미국(US)

(71) 출원인 컴퓨터 어소시에이츠 싱크, 인코포레이티드
미국, 뉴욕 11749, 아일랜디아, 원 컴퓨터 어소시에이츠 플라자
(72) 발명자 하베이, 리차드
오스트레일리아 3134 빅토리아주 링우드 오데트 코트 4
벤틀리, 티모시
미국 11749 뉴욕주 아일랜디아 원 컴퓨터 어소시에이츠플라자
(74) 대리인 주성민
정은진
백만기

심사청구 : 없음

# (54) 웹 서비스 장치 및 방법

## 명세서

### <관련 출원에 대한 상호 참조>

본 출원은 2002년 8월 26일에 각각 출원되고 그 내용이 본 명세서에 참조된 가출원 일련 번호 제60/406,391호, 제60/406,399호, 제60/406,325호, 제60/406,328호, 제60/406,204호, 제60/406,205호, 및 제60/406,319호의 이익을 주장한다.

### 기술분야

본 발명은 일반적으로 UDDI 레지스트리 및 웹 서비스에 관한 것으로서, 구체적으로는 이러한 서비스에 실질적인 효과를 주는 데 사용되는 방법, 장치 및 시스템에 관한 것이다.

### 배경기술

UDDI(Universal Description, Discovery and integration)는 웹 서비스를 이용하는 애플리케이션들이 서로 쉽게 동적으로 상호작용할 수 있도록 하기 위하여 정의된 일련의 표준들이다. UDDI는 연산 레지스트리는 물론, 서비스를 설명하고 기업을 발견하며 인터넷을 이용하여 시스템 서비스들을 통합하기 위한 플랫폼 독립식 개방 프레임워크를 생성한다. 보다 상세한 것은 웹 사이트 www.uddi.org를 참조한다.

UDDI 레지스트리는 웹 서비스를 이용하여 구축된 시스템들에 유용한 지원을 제공한다. 도 1a는 기본적인 웹 서비스 및 UDDI 개념을 개략적으로 도시하고 있다. 도 1b는 웹 서비스 환경을 위한 간단한 프로토콜 스택을 개략적으로 도시하고 있다. UDDI는 웹 서비스 정보를 위한 저장소를 제공하며, 그 자체는 웹 서비스에 의해 제공된다.

UDDI는 애플리케이션들이 웹 상에서 어떻게 상호작용하기를 원하는가를 공표할 수 있게 해준다. 각각의 '웹 서비스'는 인터넷 접속을 통해 다른 애플리케이션에 소정의 시스템 기능을 제공하는 애플리케이션 논리의 자체 설명적이고 독립적인 모듈라 유닛이다. 애플리케이션은 각각의 웹 서비스가 어떻게 구현되는지에 대해 염려할 필요 없이 유비쿼터스 웹 프로토콜 및 데이타 포맷을 통해 웹 서비스에 액세스한다. 웹 서비스는 보다 큰 작업 흐름 또는 기업 트랙잭션을 실행하기 위하여 다른 웹 서비스와 혼합되고 매칭될 수 있다.

UDDI 표준은 웹 서비스 타입, 기업 조직 및 웹 서비스가 어떻게 호출되는지에 대한 세부 사항의 설명을 관리하는 특정 목적 저장소를 설명한다. 이 표준은 표준이 어떻게 구현되어야 하는지 또는 그 구현이 데이타베이스, 디렉토리 또는 임의의 다른 매체를 이용하는 저장 장치를 포함해야 하는지를 반드시 특정하지는 않는다.

UDDI 표준을 담당하는 조직에 의해 호스트되는 웹 사이트(http://www.uddi.org/faqs.html)에는 많은 FAQ가 있다. 이들 질문 중 하나는 "UDDI 레지스트리가 LDAP 상에 구축되거나 LDAP에 기초할 수 있는가"하는 것이다. 그 대답으로, 이 웹 사이트는 UDDI와 디렉토리 사이에는 공식적인 관계가 없다고 설명한다. UDDI 사양은 레지스트리 구현 세부 사항을 지시하고 있지 않다. UDDI 사양은 XML 기반 데이타 모델 및 이 데이타 모델에 액세스하고 조작하기 위한 일련의 SOAP API를 정의하고 있다. SOAP API는 UDDI 저장소가 보이는 거동을 정의한다. UDDI 구현은 그것이 특정 거동을 따르는 한 LDAP 디렉토리 상에 구축될 수 있다. 지금까지 모든 UDDI 구현은 관계형 데이타베이스 상에 구축되어 왔다.

X.500 및 LDAP 등의 디렉토리 기술들은 사용자 및 자원을 관리하는 데 매우 자주 사용되는 확장성 범용 데이타 스토어 및 그와 관련된 언어들이라는 점에 유의한다. 이들은 매우 양호하게 구축된 기술이며, 널리 채용되고 있고 매우 안정되고 신뢰성이 높은 것으로 간주되고 있다.

그러나, 디렉토리 상에서 UDDI 표준(www.uddi.org에서 입수 가능함)을 구현하기 위해서는 많은 문제를 해결하여야 한다. UDDI 표준은 다음과 같이 해결되지 않은 많은 중요한 문제를 갖고 있다:

· UDDI 표준은 다수의 객체를 정의하며, 이들의 일부는 계층 구조에 의해 관련되지만, UDDI는 모두를 포괄하는 계층 구조를 정의하고 있지 않다. 예를 들어, 기업 서비스 객체는 기업 엔티티 객체 아래에 오고, 결합 템플릿 객체는 기업 서비스 아래에 온다. 도 2는 이러한 계층 구조의 일례를 나타낸다. 기업 엔티티 객체는 21로 표시되어 있고, 기업 서비스 객체는 22로 표시되어 있으며, 결합 템플릿 객체는 23으로 표시되어 있다. 또한, 예를 들어 24로 표시된 T모델(TModel) 객체는 상기 객체들과 계층적으로 관련되어 있지 않다는 점에 유의한다. 또한, 예를 들어 계층적으로 정의되지 않은 공표자 표명과 같은 다른 개념들도 있다.

· 사용자가 그의 제어 하에 있는 객체들만을 변경하는 것이 허용되어야 하는 요건의 효율적인 구현을 생성하는 문제.

· UDDI 레지스트리들의 분산을 허용하는 효율적인 구현을 생성하는 문제.

· 검색 및 갱신의 관리 및 성능의 양태들을 강화하는 효율적인 구현을 생성하는 문제.

· 어떻게 복잡한 UDDI 객체를 비교적 효율적인 방법으로 표현할 것인가하는 문제. 예를 들어, 기업 엔티티, 기업 서비스, 결합 템플릿 및/또는 T모델은 합성 반복 요소들을 갖는다. 또한 이들 반복 요소는 추가적인 반복 요소를 포함할 수 있다. 예를 들어, 기업 엔티티는 컨택들을 포함할 수 있고, 컨택들은 어드레스들을 포함할 수 있다. 어드레스들은 어드레스 라인들 및 전화 번호들을 포함할 수 있다. 도 13은 기업 엔티티 내의 비교적 복잡한 객체의 UDDI 개념을 개략적으로 도시한다. 기업 엔티티 객체(131)는 예를 들어 인증된 이름(AuthorizedName), 기업 키(BusinessKey) 및 이름(Name) 등과 같은 다수의 속성(132)을 포함한다. 이름은 '텍스트'와 같은 하나 이상의 이름 필드(133)를 갖거나, 이것은 '이름' 자체 내에서 암시될 수 있다. 또한 '언어'가 존재한다. 이들 필드(133) 중 하나 이상이 있을 수 있다.

· 반복 요소에 포함된 특정 항목에 대한 비교적 빠른 검색을 어떻게 제공할 것인가하는 문제.

· 디렉토리 객체들의 계층 구조에서 UDDI 정보 및 요건을 어떻게 표현할 것인가하는 문제.

· 어떻게 UDDI 객체 및 이들과 관련된 모든 정보의 삭제를 효율적인 방식으로 관리할 것인가하는 문제.

· 디렉토리 저장 매체 제한을 고려하여 디렉토리 액세스 및 반복 인-메모리 동작을 최소화하도록 검색 동작 동안 중간 검색 결과 수집의 구축을 어떻게 최적화할 것인가하는 문제. 실제로, 디렉토리 엔트리는 임의의 순서로 저장되고 반환될 수 있으며, 디렉토리 결과는 너무 커서 분류할 수 없을 수 있다.

· 어떻게 공표자 표명에 관한 데이타를 효율적인 방법으로 표현할 것인가하는 문제.

· 관련 기업 발견(findrelatedBusiness) 메서드의 구현과 특히 관련하여 공표자 표명의 효율적인 구현을 어떻게 생성할 것인가하는 문제.

· 관계에 의해 공표자 표명의 효율적인 검색을 어떻게 구현할 것인가하는 문제.

· 공표자 표명의 유효성을 어떻게 관리할 것인가하는 문제.

· 기업 엔티티에 대해 생성되고 삭제되는 표명이 기업 엔티티의 소유자에 의해 만들어지는 것을 어떻게 제한할 것인지하는 문제.

· UDDI 표준에서 정의되는 바와 같이 속성들의 관련 수집을 어떻게 효율적으로 관리할 것인가하는 문제.

· 검색의 성능을 강화하기 위하여 어떻게 속성 및 객체를 정의할 것인가하는 문제.

다양한 UDDI 스키마가 제안되어 왔다. 그러나, 그 어느 것도 적어도 전술한 문제들을 해결할 수 있는 것으로 고려되고 있지 않다. 예를 들어, 한 스키마는 효율적인 상업적 구현을 이루기 위해 복잡성 및 최적화를 고려할 필요 없이 UDDI 객체들의 디렉토리 객체들로의 비교적 간단한 맵핑을 제공한다. 이러한 스키마에서도 어떻게 많은 UDDI 서비스(특히 find_series)를 구현할 수 있을지는 불명확하다.

예를 들어, 도 14는 기업 엔티티 내의 비교적 복잡한 객체의 노벨(Novell) 표현을 개략적으로 도시하고 있다. 기업 엔티티 객체(141)는 예를 들어 '타입(type)" 및 '값(value)'을 각각 갖는 다수의 속성(142)을 포함한다. 도시된 바와 같이, 값 'Bill'을 가진 인증된 이름, 값 '890.obale.890.....'을 가진 기업 키, 및 다수의 값(143, 144), 즉

en#CA

IN#CATS

를 가진 이름이 있다.

UDDI(도 13) 및 노벨(도 14)의 예시적인 표현은 웹 서비스에 대한 효율적인 표현인 것으로 간주되지 않는다.

따라서, 디렉토리에 기초하여 비교적 확장 가능하고 효율적이며 신뢰성 있는 UDDI의 구현을 제공하기 위하여 전술한 일반적인 문제 및 다른 문제들을 해결할 필요가 있다.

## 발명의 상세한 설명

사용자 객체 아래에 기업 엔티티 객체를 배열하고, 사용자 객체, 저장소 객체 및 접두사 중 적어도 하나에 상응하는 T모델 객체를 배열하는 단계를 포함하는 웹 서비스 장치에서 이용되는 방법에 관한 것이다.

## 도면의 간단한 설명

본 발명의 추가적인 목적, 이점 및 양태들은 첨부된 도면들을 참조한 아래의 바람직한 실시예들의 설명으로부터 보다 잘 이해될 수 있다.

도 1a는 소정의 웹 서비스 및 UDDI 개념들을 나타내는 개략도이다.

도 1b는 웹 서비스 환경을 위한 간단한 프로토콜 스택의 개략도이다.

도 2는 종래 기술에 따른 계층 구조의 개략도이다.

도 3은 종래 기술에 따른 디렉토리 서비스 모델의 개략도이다.

도 4는 본 발명의 일 실시예에 따라 X.500 디렉토리 기술을 이용하여 구현되는 UDDI 서비스 모델에 대한 기반구조 컴포넌트들의 개략도이다.

도 5는 본 발명의 일 실시예에 따른 서비스 프로젝션의 개략도이다.

도 6은 본 발명의 일 실시예에 따른 결합 템플릿과 T모델 사이의 관계를 나타내는 개략도이다.

도 7은 본 발명의 일 실시예에 따른 2개의 엔티티 사이의 관계를 T모델이 어떻게 생성하는지를 나타내는 도면이다.

도 8은 본 발명의 일 실시예에 따른 공표자 표명을 추가하기 위한 요구의 논리적 표현을 나타내는 도면이다.

도 9는 본 발명의 일 실시예에 따른 UDDI 데이타 객체에 대한 구축자의 논리적 표현을 나타내는 도면이다.

도 10은 사용자 객체 아래에 기업 엔티티 객체를 배치하는 것을 개략적으로 나타내는 도면이다.

도 11은 사용자 객체 위에 도메인 객체를 배치하는 것을 개략적으로 나타내는 도면이다.

도 12는 본 발명의 일 실시예에 따른 스키마의 개요를 개략적으로 나타내는 도면이다.

도 13은 종래 기술에 따른 기업 엔티티 내의 비교적 복잡한 객체의 UDDI 개념을 개략적으로 나타내는 도면이다.

도 14는 기업 엔티티 내의 비교적 복잡한 객체의 노벨 표현을 개략적으로 나타내는 도면이다.

도 15는 기업 엔티티 내의 비교적 복잡한 객체의 표현에 대한 본 발명의 일 실시예에 따른 계층 구조의 도입을 개략적으로 나타내는 도면이다.

도 16은 본 발명의 일 실시예에 따른 결합 템플릿 계층 하위 구조를 개략적으로 나타내는 도면이다.

도 17은 단순화 및/또는 병합된 결합 템플릿 하위 구조를 개략적으로 나타내는 도면이다.

도 18은 본 발명의 다양한 양태들을 구현할 수 있는 컴퓨터 시스템의 블록도이다.

## 실시예

도면에 도시된 본 발명의 바람직한 실시예들을 설명하는 데 있어서, 간략화를 위해 특정 용어가 사용된다. 그러나, 본 발명은 선택된 특정 용어에 한정되는 것은 아니며, 각각의 특정 요소는 유사한 방식으로 동작하는 모든 기술적 균등물을 포함하는 것으로 이해되어야 한다.

도 18은 본 발명의 방법 및 시스템을 구현할 수 있는 컴퓨터 시스템의 일례를 나타낸다. 본 발명의 시스템 및 방법은 예를 들어 메인프레임, 개인용 컴퓨터(PC), 핸드헬드 컴퓨터, 서버 등의 컴퓨터 시스템 상에서 실행되는 소프트웨어 애플리케이션의 형태로 구현될 수 있다. 소프트웨어 애플리케이션은 컴퓨터 시스템에 의해 로컬 액세스될 수 있는 기록 매체, 예를 들어 플로피 디스크, 컴팩트 디스크, 하드 디스크 등에 저장되거나, 컴퓨터 시스템으로부터 멀리 떨어져 있어 예를 들어 근거리 통신망 또는 인터넷과 같은 통신망에 대한 유무선 접속을 통해 액세스될 수 있다.

본 발명의 방법 및 시스템을 구현할 수 있는 컴퓨터 시스템의 일례가 도 18에 도시되어 있다. 일반적으로 시스템(180)으로 참조되는 컴퓨터 시스템은 중앙 처리 장치(CPU)(182), 예를 들어 랜덤 액세스 메모리(RAM)와 같은 메모리(184), 프린터 인터페이스(186), 표시 장치(188), LAN 데이타 전송 제어기(190), LAN 인터페이스(192), 망 제어기(194), 내부 버스(196), 및 예를 들어 키보드, 마우스 등의 하나 이상의 입력 장치(198)를 포함할 수 있다. 도시된 바와 같이, 시스템(180)은 링크(202)를 통해 예를 들어 하드 디스크(200)와 같은 데이타 저장 장치에 접속될 수 있다.

다음은 본 발명의 실시예들의 현저한 특징들의 일부 및 이들에 의해 제공되는 몇몇 이점들을 요약한 것이다.

본 발명의 일 실시예에 따르면, 사용자 위에 저장소 계층이 생성되므로 각각의 저장소는 다른 서버 상에 배치될 수 있다. 이러한 저장소 계층은 디렉토리 접두사를 집합적으로 형성하는 하나 이상의 디렉토리 노드를 포함한다. 이것은 저장소의 도메인 또는 이름으로도 알려져 있을 수 있다. 이것의 이점은 도메인에 대한 정보를 유지할 단일 장소를 제공한다는 점이다. 이 노드의 이름은 디렉토리 접두사를 나타낸다.

UDDI 계정을 나타내는 데이타를 유지하기 위하여 사용자 객체가 생성될 수 있다. 이것의 이점은 사용자/계정에 대한 정보를 유지할 단일 장소를 제공한다는 점이다.

기업 엔티티 객체는 사용자 객체 아래에 배열될 수 있고, 기업 서비스 객체는 기업 엔티티 객체 아래에, 그리고 결합 템플릿 객체는 기업 서비스 객체 아래에 배열될 수 있다. 이것의 이점은 사용자 객체 계층 위의 저장소 또는 도메인 계층이 다수의 저장소가 함께 배치되거나 로컬 접속될 수 있게 해준다는 점이다. 도메인 계층은 예를 들어 대륙별로 구성된 여러 나라들, AU, US, EP 등을 가진 다수의 레벨로 배열될 수 있다.

또 하나의 이점은 이러한 기능이 X.500 디렉토리의 분산 기능들을 이용하여 효과를 얻을 수 있다는 점이다. 예를 들어, 이것을 구현하기 위하여, 월드(World) 또는 회사(Corporation) 노드는 가상 디렉토리 트리의 최상부에 배치되며, 고유하게 명명된 노드가 각각의 UDDI 서브 트리(UDDI 이름 공간)의 최상부에 배치된다. 이들 노드 접두사는 사용자에게는 보이지 않지만 UDDI 저장소가 디렉토리 분산을 쉽게 할 수 있도록 해준다.

본 발명의 일 실시예에 따르면, 기업 엔티티 객체는 사용자 객체의 자식으로 간주될 수 있다. 기업 엔티티, 기업 서비스 및 결합 템플릿 계층 구조 상에 사용자/계정을 갖는 것은 각각의 사용자가 그들 자신의 서브 트리를 갖는 효과를 제공한다. 이것은 관리성 및 보안성을 향상시킨다. 사용자는 그 자신의 서브 트리만을 수정 및/또는 관리하도록 쉽게 제한된다. 이것은 또한 디렉토리 서브 트리 검색 동작을 이용함으로써 성능을 향상시킨다.

일 실시예에 따르면, 사용자에 의해 정의되는 T모델은 사용자 객체의 자식으로 간주될 수 있으며, 따라서 보안을 구현하기 쉽게 만든다. 이것은 사용자가 그 자신의 서브 트리를 수정 및/또는 제어할 수 있게만 하므로 관리성 및 보안성을 향상시킨다. 또한, 이것은 디렉토리 서브 트리 검색 동작을 이용함으로써 성능을 향상시킨다.

본 발명의 일 실시예는 X.500/LDAP 디렉토리 기술을 이용하여 UDDO 환경의 맵핑을 나타낸다. 특히, X.500 및 LDAP 디렉토리 기술의 계층 구조는 UDDI 환경에 적당한 것으로 밝혀졌다. 추가 요소들(예를 들어 사용자 객체)의 주의 깊은 설계는 계층 구조를 UDDI 환경의 요건에 보다 더 적당하게 만들었다.

본 명세서의 전반에서, '디렉토리'라는 용어는 X.500, LDAP 및 유사한 기술을 포함하며, '사용자'라는 용어는 '계정'도 포함하는 것으로 이해되고, 그 역도 마찬가지며, '저장소'라는 용어는 '디렉토리 접두사', '도메인' 또는 '노드'도 포함하는 것으로 이해되고, 그 역도 마찬가지다.

웹 서비스는 원래 조직들, 예를 들어 기업들, 파트너들, 고객들, 공급자들 사이의 서비스인 것으로 생각되었었다. 이러한 관점에서, UDDI는 이들 조직이 제공하는 서비스를 위한 단일 저장소로서 생각되었었다.

이제, 웹 서비스 및 UDDI가 기업 내에서 조직 내부의 애플리케이션들을 통합하는 데 유용하다는 것은 명백하다. 웹 서비스 및 UDDI가 소정 벤더로부터의 제품 세트 내의 제품들을 통합하는 데 이용될 수 있음도 명백하다. 상업적인 환경 외에 정부 부처, 대형 교육 기관, 및 많은 다른 비영리 단체들과 같은 분야에서도 이용될 수 있다.

아래의 설명은 기업과 관련하여 설명되지만 임의의 다른 환경에도 동일하게 적용될 수 있으며, 전술한 타입의 환경에 특히 적용될 수 있다.

기업 UDDI 레지스트리는 기업 내에서 내부 소비용 정보 및 서비스를 공표하도록 전개될 수 있는 서비스일 수 있다. 또한, 기업 UDDI 레지스트리는 분산 애플리케이션에 대한 구성 발견과 같은 다른 기능을 제공하는 데 도움이 될 수 있다.

웹 서비스는 내부적으로, 그리고 파트너들과 함께 기업 프로세스들을 빠르고 쉽게 통합하려는 소망에 의해 이루어지고 있다. 웹 서비스를 효율적으로 이용하는 하나의 컴포넌트는 소프트웨어 컴포넌트들이 인터넷을 통해 적당한 서비스를 동적으로 발견하고 접속할 수 있게 하는 공개 UDDI 레지스트리이다. 웹 서비스는 또한 기업 내의 기업 프로세스들의 통합을 약속한다. 이 경우, UDDI 레지스트리는 조직의 기반 구조의 일부(예를 들어 중요 기업 애플리케이션)가 될 수 있으며, 따라서 가장 높은 레벨의 보안성, 성능, 신뢰도 및 관리성을 제공한다. 디렉토리 기술은 기업 UDDI 레지스트리의 엄격한 요건을 지원하는 이상적인 토대를 제공한다.

기업 UDDI 레지스트리는 UDDI에 대한 표준 지원을 전달하는 레지스트리로서 정의될 수 있지만, 전개를 위해 4개 영역을 어드레스할 수 있도록 그것을 초월한다. 이들 영역은 액세스를 인증된 사용자로만 제한하는 보안성, 대규모 전개를 지원하는 분산성, 진정한 생산 시스템을 위한 관리성, 및 서비스 레벨 약정을 만족시키는 이용성을 포함한다.

강력한 보안성은 소정 기업 전개를 위한 중요한 요건일 수 있다. 공개 UDDI 레지스트리는 어느 누구나가 이용 가능 서비스를 발견하는 것을 돕고자 하는 유일한 목적을 위해 존재한다. UDDI 레지스트리는 정당한 사람들이 이들 서비스를 발견할 수 있게 하기 위한 유일한 목적을 위해 존재한다. 이것은 중요한 차이이다.

인터넷 UDDI 레지스트리는 기업 내에서 웹 서비스를 전개하는 데에 부적합한 것으로 간주되고 있다. 예를 들어, 페이롤 시스템 또는 피고용자의 이익 관리 애플리케이션에 인터페이스하는 웹 서비스의 정의들은 인터넷 UDDI 레지스트리로 전송되지 않는다.

또한, 보안 요건은 내부적으로 전개되는 UDDI 레지스트리도 강력한 액세스 제어를 제공해야 한다는 것을 의미할 수 있다. 이것은 UDDI 레지스트리가 본질적으로 무엇이 행해질 수 있고 그것을 어떻게 행할 것인지에 대한 지침을 제공하기 때문이다. UDDI 레지스트리는 임의의 이용 가능한 웹 서비스들에 대한 기업 레벨의 설명, 및 이들 서비스에 대한 프로그램적인 인터페이스를 완전하게 정의하는 WSDL에 대한 지침을 제공한다. 이것은 애플리케이션 개발자는 물론 해커에게도 높은 생산성의 도구를 제공한다.

따라서, 재정적으로 민감하거나 비밀스러운(의료 기록 등) 시스템에 대한 인터페이스 정의들로의 액세스를 제한하는 것이 바람직하다. 전개 조직 내부에서도 특정 웹 서비스에 대한 정보로의 액세스를 인증된 사람들로만 제한하는 것이 현명할 수 있다.

기업 내에서 또는 선택된 사업 파트너들과 함께 엑스트라넷을 통해 비보안 UDDI 레지스트리를 사용하는 것은 지극히 위험할 수 있다. 무료로 다운로드될 수 있는 도구들 덕분에, 비교적 낮은 레벨의 전문 기술을 가진 사람들도 웹 서비스에 액세스하여 이를 이용할 수 있다. 임의의 진정한 기업 솔루션은 웹 서비스에 대한 정보로의 액세스를 투명하게 제어할 수 있는 능력을 갖춘 표준 UDDI 서비스를 구현할 수 있다.

분산과 관련하여, 많은 경우에, UDDI 레지스트리의 초기 전개는 소규모일 것이다. 그러나, 웹 서비스 요구가 증가하면, 대규모 전개가 보다 일반적으로 될 것이다. 또한, 레지스트리 이용 및 전개는 UDDI 레지스트리에 대한 새로운 기능의 발견과 더불어 가속화될 것이다.

보다 큰 구현 및 지리적으로 분산된 조직들 내에서의 이용은 단일 조직 내에서 다수의 UDDI 레지스트리의 구현을 이끌 것이다. 분산 레지스트리로의 발전은 임의의 개별 레지스트리가 다른 레지스트리들과 동적으로 상호작용하여 그들의 요구를 서비스할 수 있는 것을 중요하게 만든다. 내부 레지스트리 통신이 설정되면, 이는 믿을만한 사업 파트너들의 레지스트리들 또는 인터넷 UDDI 레지스트리들도 포함하도록 방화벽을 넘어 확장될 수 있다.

내부 레지스트리 통신에 대한 필요를 해결하기 위한 2가지 기본 방법이 있는 것으로 고려된다. 하나의 방법은 동일한 엔트리 이름 공간이 다수의 서버 상에 존재하는 '복제'이다. 또 하나의 방법은 상호 접속된 서버들이 상이한 엔트리 이름 공간을 갖지만 하나의 논리적 서비스로서 동작하는 '분산'이다.

이들 두 방법은 종종 유사한 것으로 혼동될 수 있지만, 이들은 아주 다르다.

복제 방법에 있어서, 정보는 그것을 탐색할 필요가 있는 모든 서버에서 복제된다. 이것은 비교적 단순하고, 심지어 극단적으로 단순한 솔루션이지만, 갱신들을 동기화해야 하는 요건을 수반하며, 정의에 의해 레지스트리들의 수 및 그들의 내용들의 양이 증가함에 따라 네트워크 혼잡성을 증가시킨다. 복제 기술은 서버 수가 적고 정보량이 적으며 변화가 빈번하지 않은 환경에 최적이다. 기업 전개에 있어서, 복제는 페일오버 환경에서 백업 저장소를 유지하는 데 가장 유용하다. 복제 기술을 이용하여 지리적으로 또는 기능적으로 분산된 서버들을 동기 상태로 유지하는 것은 매우 어렵다.

분산 방법에 있어서, 정보는 각각의 참여 서버 상에서 논리적으로 표현되나, 단일 레지스트리에만 저장된다. 필요할 때만 질의들이 다른 레지스트리들로 분산된다. 따라서, 반환된 정보는 통용되고 있는 정보인 것으로 보증된다. 이것은 단일 갱신 시점을 제공하며, 복제 기술에 내재되어 있는 동기화 및 대역폭 소모의 문제를 일소한다. 진정한 분산은 서버들 사이의 스케일 가능한 접속성에 대한 하나의 해답인 것으로 고려되고 있다.

기업 UDDI 레지스트리에 있어서, 분산이 일반적으로 이용되는 2가지 시나리오가 있다. 첫째는 각각이 새로운 UDDI 엔트리를 생성하고 UDDI 서비스를 소비하는 지리적으로 분리된 사무실들을 갖춘 조직들에 대한 것이다. 단일의 중앙화된 UDDI 레지스트리를 실행하는 것이 가능할 수 있지만, 대역폭 제한 및 시간 대역 차이는 자주 동작이 불가능한 시점에 대하여 분산을 어렵게 만든다.

분산 레지스트리는 유연하고 스케일 가능한 솔루션을 제공한다. 이 시나리오에서, 각각의 참여 사무실은 개별 레지스트리를 가지며, 각 레지스트리는 다른 레지스트리를 그 자신의 내용의 논리적 부분으로 본다. 레지스트리 서비스는 모든 접속 세부 사항을 관리하며, 고객들은 지리적 여건을 걱정할 필요가 없게 된다.

제2 시나리오는 기업이 그의 내부 UDDI 시스템을 믿을만한 파트너의 시스템 또는 공개 인터넷 레지스트리에 접속시키는 것이 필요할 때 발생한다. 특히 공개 레지스트리의 경우, 복제는 문제가 있다. 인터넷 레지스트리 오퍼레이터들은 그들의 레지스트리의 일부가 기업의 내부 레지스트리로 복제되는 것을 바라지 않을 수 있다. 역시, 분산 방법이 하나의 해답이다. 현재, 분산에 대한 UDDI 표준이 없으며, 복제에 대한 제안들은 복잡한 것으로 간주되고 있다. 하나의 솔루션은 표준에 대한 수정을 요구하지 않고 UDDI 분산 방법의 이익을 제공한다.

관리성과 관련하여, 기업 내에서 임무에 중대한 기능들을 수행하는 컴포넌트로서, UDDI는 성능 및 신뢰도 요건을 만족시켜야 한다. 이것은 개발자에게 편리한 유틸리티만으로 존재해서는 안된다. 클라이언트들에 의한 판독 액세스는 가장 빈번하고 가장 시간적으로 중요한 UDDI 레지스트리의 이용일 것이다. 성능은 최대 처리량을 위해 최적화되며, 탐색 질의들에 대한 응답 시간은 보다 복잡한 검색에 의해 영향을 받아서는 안된다. 레지스트리의 크기 및 복잡성이 증가함에 따라 성능이 저하되면 안된다. UDDI 레지스트리를 지원하는 데이타 스토어는 산업적인 강도이어야 하며, 트랜잭션 및 자동 복원을 완전히 지원해야 한다. 또한, UDDI 서버들은 높은 이용도를 가져야 하며, 네트워크 페일오버 및 핫 스탠바이(hot standby)와 같은 기능을 지원해야 한다. 시스템 관리자들은 UDDI 레지스트리를 유지하고 모니터하고 제어하기 쉽게 만들 수 있는 능력을 가져야 한다. 이러한 능력은 서비스를 오프라인 상태로 하지 않고 제어, 규칙 및 설정을 변경할 수 있는 '동적 구성', 높은 이용성을 위한 '온라인 백업 및 튜닝', 레지스트리의 "트롤링(trawling)"을 중지하고 서비스 거부(denial-of-service) 공격을 방지하는 '관리 제어', 'SNMP를 통한 모니터링' 또는 다른 타입의 경고 메카니즘, 보안, 통계, 질의 및 갱신 정보에 대한 개별 로그 파일을 구비한 '감사 및 진단', 및 복제, 분산 및 라우팅을 지원하는 '전개' 옵션을 포함한다.

개발자에 초점이 맞춰진 많은 UDDI 레지스트리들이 도입되어 왔다. 이들은 작은 개발 팀에게는 유용한 성능을 제공하지만, 진정한 생산 품질 시스템은 아니다. 웹 서비스 전개는 빠르게 성장하고 있으며, 이에 대응하여 진행 웹 서비스 전개를 지원하도록 빠르게 스케일될 수 있는 기업 품질 레지스트리가 요구되고 있다.

UDDI 레지스트리는 하나의 서비스를 제공한다. 이 서비스는 많은 애플리케이션들에 의해 의지될 것이다. 온라인 사업의 경우, 이러한 서비스가 항상 존재하는 것이 중요할 수 있다. 예를 들어, UDDI 레지스트리는 99.99% 이용도의 서비스 레벨 약정을 제공하도록 요구될 수 있다. 이러한 레벨의 이용도를 가능하게 하기 위하여, UDDI 레지스트리는 둘 이상의 기계에서 복제될 수 있으며, 및 이 기계들을 동기 상태로 유지하고 이 기계들 중 하나를 이용할 수 없더라도 임의의 수신 질의들을 이용 가능한 기계로 자동적으로 라우팅하는 것을 보장하는 메카니즘들이 제공될 수 있다.

지시된 바와 같이, UDDI는 전화 디렉토리 서비스와 실제로 유사한 것으로 간주될 수 있다. 따라서, 정보 저장의 디렉토리 모델은 UDDI 레지스트리 서비스를 구축할 수 있는 완벽한 기반이다. 디렉토리 모델은 디렉토리 기반 서비스에 대한 특정 요구에 대해 기업 레벨 전개에 필요한 보안성, 스케일 가능성 및 신뢰성을 갖추도록 발전되고 개발되어 왔다.

전술한 항목들의 대부분은 애플리케이션 아키텍쳐에서 데이타 저장 레벨이 아니라 서비스 레벨에서 구현된다. 관계형 데이타베이스(RDBMS)는 많은 다른 종류의 애플리케이션들이 구축될 수 있는 일반적인 툴 키트이다. RDBMS 구현은 엔드 애플리케이션에서 요구되는 추가 서비스 기능이 아니라 실질적인 데이타 액세스 기능을 제공하는 것에 집중되고 있다.

도 3에 도시된 디렉토리 서비스 아키텍쳐는 다른 컴포넌트들로부터의 서비스 계층(31)의 분리를 도시하고 있다. 인터페이스 기능들을 서비스 계층(31) 내에 캡슐화하는 것은 서비스 기반 구조 컴포넌트들이 재사용될 수 있게 한다. 웹 서버가 이에 대한 뛰어난 예이다. 웹 서버는 독립 컴포넌트로 구축되기에 충분히 유용한 서비스를 함께 구성하는 일련의 기능들(HTTP 액세스, CGI 프로세싱 등)을 제공한다. 같은 방식으로, 디렉토리 서비스 모델은 특정 타입의 애플리케이션에 의해 요구되는 기능들을 제공하도록 개발되어 왔다. 디렉토리 기술들은 인증 및 허가 분야에서 임무에 중요한 많은 애플리케이션들에 대한 지원을 제공한다.

UDDI는 다른 종류의 디렉토리 서비스와 유사한 것으로 보일 수 있다. 게다가, UDDI에 의해 제기되는 구현 문제들의 대부분이 디렉토리 기술들을 이용하여 해결될 수 있는 것으로 보여질 수 있다. 예를 들어, 디렉토리들은 UDDI 전화 디렉토리 동작들에 대해 매우 공통적인 극히 효율적인 발견 및 검색 동작들을 위해 최적화된다.

UDDI 서비스가 기업 내에서 성공적으로 전개될 경우 강력한 보안, 분산 및 관리 능력을 제공해야 한다는 점은 이미 주목되어 왔다. 기업 강도 디렉토리 서비스 솔루션들 내에 이미 구축되어 있는 바로 동일한 속성들이 존재한다.

기업 UDDI 레지스트리를 구축하기 위한 하나의 방법은 고성능의 실제 애플리케이션들에서 시도되고 테스트되어 왔던 기존의 디렉토리 기반 구조를 확장하는 것이다.

디렉토리 서비스 아키텍쳐는 기업 UDDI 레지스트리를 구현할 수 있는 최적의 수단을 제공한다. 이러한 조합은 성공에 필요한 능력을 지원한다. 도 4에 개략적으로 도시된 UDDI 서비스는 이러한 기반 구조에 대해 구현될 수 있는 컴포넌트들을 식별하고 있다. UDDI 시맨틱 브리지(41)는 UDDI의 SOAP 구현(42)과 디렉토리(44)에 의해 지원되는 LADP 인터페이스(43) 사이를 중개하는 서비스 컴포넌트이다. 디렉토리(44)는 보안 제어, 분산 메카니즘 및 관리 능력을 지원하는 것과 함께 정보 액세스를 전달한다. RDBMS(45)는 기초적인 물리 데이타 관리, 트랜잭션 완전성 및 백업 및 복원 메카니즘을 제공한다.

UDDI 레지스트리 제품들은 RDBMS 기술 상에 직접 구축될 수 있다. 관계형 데이타베이스는 많은 방법에서 매우 유용하고 강력하지만 그 자체만으로는 디렉토리 프로세싱에 고유한 요건을 만족시키지 못한다.

하부에 RDBMS 또는 다른 데이타 저장 시스템을 이용하여 스크래치로부터 디렉토리 타입 애플리케이션을 구축할 수 있다. 그러나, 이것은 가장 효율적인 방법이 아닐 수 있다.

대안적인 하나의 방법은 디렉토리 서비스 모델을 이용하여 UDDI 레지스트리를 전달하고 이러한 특정 타입의 애플리케이션에 필요한 기능들을 제공하는 것이다. UDDI 레지스트리에 필요한 보다 많은 기능들은 현대적인 산업 강도 디렉토리 서비스들에 의해 제공될 수 있다. UDDI 레지스트리는 특수 통신 및 API들을 갖춘 디렉토리 서비스로 보일 수 있다. 디렉토리 상의 UDDI 서비스 전달은 이익을 얻기 위해 UDDI 표준을 수정할 필요 없이 필요한 보안, 분산 및 관리 능력을 제공할 수 있다.

데이타 표현의 세심한 설계는 UDDI 저장소에 필요한 기능 및 성능을 제공하는 데 이롭다.

아래의 설명은 다양한 UDDI 개념을 참조한다. 이러한 UDDI 개념들에 대한 보다 상세한 설명은 UDDI 사양들(http://www.uddi.org/specification.html)을 참조하여 얻어질 수 있다.

디렉토리 용어에서 스키마는 디렉토리에 저장될 수 있는 데이타 요소들, 및 이들 요소들이 어떻게 함께 연결될 수 있는지에 대한 설명이다. 이것은 가능한 속성들(속성은 데이타의 단일 부분을 유지한다) 각각에 대한 설명, 다양한 객체들(객체는 속성들의 집합이다)에 대한 설명, 및 가능한 객체 계층 구조들의 사양을 포함한다. 본 명세서에 사용되는 특정 스키마 표기법은 컴퓨터 어쏘시에츠 인터내셔널 사의 제품인 eTrust 디렉토리에 의해 사용되는 것이다. 'eTrust'는 컴퓨터 어쏘시에이츠 인터내셔널 사의 제품명이자 상표이다. 물론, 다른 스키마 표기법들도 사용될 수 있다.

본 명세서는 디렉토리를 데이타 스토어로서 이용하여 UDDI 저장소를 구현하는 데 이용되는 스키마를 설명한다. 이러한 스키마에 수반되는 많은 개념들이 있다. 또한, UDDI 구현의 동작을 향상시키는 데 이용되는 많은 기술이 있다. 다음은 이러한 개념들의 일부에 대한 간단한 설명이다. 이러한 개념들 및 기술들에 대한 보다 상세한 설명은 본 발명의 실시예들을 설명할 때 후술될 것이다.

본 스키마는 최적 동작을 제공하도록 설계된다. 속성들, 객체 클래스들, 엔트리들 및 계층 구조의 정의를 포함하는 본 스키마 설계는 동작들을 향상시키는 방식으로 구현된다. 본 스키마 설계는 적어도 보안, 성능, 관리성 및 분산에 있어서 커다란 이점을 제공한다.

이제, 시스템의 계층 구조가 설명된다. X.500 디렉토리는 내부적으로 분산을 지원하며, UDDI 레벨에서의 어떠한 코딩도 없이 분산 UDDI 저장소를 제공한다. 하나의 레벨이 저장소의 내용들을 분할한다. (선택적인) 이 스키마의 도메인 레벨은 레벨, 각각의 도메인 엔트리, 및 그 아래의 엔트리들 모두가 UDDI 레벨 프로그래밍에 투명하게 개별 디렉토리 서버 상에 배치될 수 있음을 규정한다. 도 11은 이러한 본 발명의 양태의 일 실시예를 나타낸다. 이것은 아래에 보다 상세히 설명된다.

본 발명의 일 실시예에 따르면, 사용자 객체는 기업 및 T모델 객체 상에 배치된다. 사용자 객체는 사용자와 관련된 정보의 저장을 위한 장소를 제공한다. 이것은 또한 사용자에 의해 공표되는 모든 데이타에 대한 지지점을 제공한다. 도 10은 이러한 본 발명의 양태의 일 실시예를 나타낸다. 이는 이하에서 더욱 상세히 설명된다.

보안은 이러한 도메인/사용자 계층 구조 시스템에서 용이하게 된다. UDDI 구현은 사용자가 그들의 데이타 객체들의 서브트리에 대한 제어를 갖도록 할 수 있다.

사용자 제어 엔트리에 대한 검색이 제공된다. 이러한 사용자에 의해 제어되는 데이타의 검색은 사용자 객체 아래의 서브트리 검색을 이용하여 향상될 수 있다.

예를 들어, 결합 템플릿에서 발생하는 T모델을 지정함으로써 기업을 발견할 수 있다. 이것은 그 자식들 중 하나(또는 그 이상)를 발견함으로써 x를 찾는 것과 같다. 즉, 질의는 T모델을 참조하는 결합 템플릿을 가진 서비스를 갖는 모든 기업을 찾을 수 있다. 이러한 질의들은 후손 객체의 DN(현저한 이름)을 발견하고 원하지 않는 레벨들을 폐기하여 기업 엔티티의 DN을 산출함으로써 행해진다. 이러한 방식으로 사본을 제거하는 것도 가능하다. 이러한 발견 기능은 본 발명의 구조의 계층적 특성으로 인해 나온 것이다.

검색은 객체 클래스에 고유한 속성들을 이용하여 수행될 수 있다. 이것은 2가지 이점을 가진 최적화이다. 이것은 검색의 기록을 단순화하며, '약한' 절들의 제거를 통해 우수한 성능을 산출한다. '약한' 절은 많은 수의 엔트리를 반환하거나 많은 엔트리들의 일부인 속성을 참조하는 필터의 일부이다. 모든 이름에 대해 동일한 속성 이름을 사용하는 설계는 이름에 의해 기업 엔티티를 검색할 때 2가지 선택을 갖게 되는데, 이것은 검색에 객체 클래스를 포함시키거나 검색의 결과들을 필터링한다. 전자는 기업 이름들이 고유한 객체 클래스를 가졌지만 객체 클래스가 약한 절이어서 보다 많은 오버헤드를 초래하는 경우에만 가능하다. 후자는 추가 코드 및 원하는 결과보다 훨씬 큰 결과 리스트를 반환할 잠재성을 의미한다.

예를 들어, 이름에 "McKenna's"를 모두 포함하는 광범위한 웹 서비스들을 제공하는 "McKenna's Testing Services"라는 회사를 고려하면, 이름에 "McKenna's"를 가진 기업 엔티티들에 대한 검색은 서비스들 모두에 대해 똑같이 중간 결과를 반환할 것이다. 이러한 중간 결과들은 제거될 수 있지만, 이들의 처리는 성능을 저하시킨다.

검색에서 속성 이름을 지정하고, 그 속성 이름이 찾고 있는 객체 클래스를 고유하게 식별하도록 하는 것이 바람직하다. 위의 예에서, 검색은 (euBusinessEntityName=McKenna's*)를 지정할 경우 보다 간단해진다.

이러한 설계는 원하는 영역만을 검색하기 때문에 효율적인 강력한 검색을 제공한다. 강력한 검색은 적은 수의 엔트리를 반환하는 검색을 포함한다. 디렉토리는 euBusinessEntityName 속성을 인덱스하고 그 인덱스로부터 결과를 반환하는데, 이것은 양호한 성능을 제공하며, 불필요한 중간 결과들을 처리하는 것을 피하게 한다.

간단한 질의에 대해, 이러한 설계는 기업 엔티티 이름에 대한 검색이 다른 설계에서 필요할 수 있는 복합 절이 아니라 단일 절이라는 것을 의미한다. 이름 속성이 euName이고, 기업 엔티티 이름이 euBusinessEntityName이라고 가정하자. 이것은 다음과 같은 검색을 산출할 것이다:

(&(euName=McKenna's*)(oc=euBusinessEntityName))

모든 이름들이 동일한 객체 클래스에 저장되는 보다 더 간단한 설계가 있다. 이것은 검색이 다시 (euName=McKenna's*)로 줄여지는 것을 의미하지만, 이제 우리는 모든 이름들에 대한 결과를 모두 읽어 기업 엔티티인 부모 객체를 가진 것들을 찾아야 하는데, 이러한 설계는 잠재적으로 열악한 성능 및 보다 복잡한 프로그래밍을 산출한다.

사례 민감도를 위해 쉐도우 속성이 사용될 수 있다. 이것은 단일 인덱스를 이용하여 사례에 민감한 검색 및 사례에 민감하지 않은 검색을 제공하기 위한 시도와는 거리가 멀다. 하나의 옵션은 결과들을 사례에 민감하지 않게 인덱스한 후 사례에 민감하게 스캐닝하는 것이다. 또 하나의 솔루션은 오리지날 데이타를 사례에 민감하게 인덱스하고, 사례에 민감하지 않게 인덱스된 제2 속성(이 안에 동일 데이타가 저장됨)을 추가하는 것이다. 이어서, 필요한 모든 것은 발견 한정사에 따라 검색할 적당한 속성을 선택하는 것이다.

이러한 설계에서 모든 속성은 단일 값을 가질 수 있다. 이것은 효율적인 인덱싱, 보다 뛰어난 성능 및 보다 강력한 검색을 가능하게 한다.

다가 속성의 사용은 모호한 검색을 유발할 수 있다. 즉, 직관에 반하고 의도되지 않은 검색 결과를 얻게 될 수 있다. 'n'이라고 하는 다가 수치 속성, 및 2와 5의 값을 가진 이 속성을 포함하는 엔트리를 고려하면, 이 엔트리는 쉽게 예상되는 것이 아닌 검색 (&(n<3)(n>4))에 응답하여 반환될 것이다.

단일 값을 가진 속성들은 강력한 검색에 이용되는 기술들 중 하나이다. 강력한 검색은 인덱스를 통해 후보 결과들의 대부분을 제거할 수 있는 검색이다. 강력한 검색은 성능 향상의 열쇠이다.

서비스 투영을 위해 별명이 사용될 수 있다. 이것은 X.500 디렉토리를 데이타 스토어로서 사용하는 커다란 이익이다. 서비스 투영은 X.500 별명을 이용하여 깔끔하게 표현될 수 있다. 이것은 데이타 완전성을 보증하는 주요 이점을 갖는다. 별명은 오리지날 데이타를 액세스하며, 따라서 오리지날에 대한 임의의 변화가 별명에 의해 즉시 반사된다. 디렉토리 구현이 별명 완전성을 지원하는 경우, 오리지날 엔트리가 삭제될 때, 별명은 추가 작업 없이 소멸된다.

공표자 표명은 UDDI 표준에서 명료하게 정의되는 최소한의 요소들 중 하나이며, 이것은 세심한 설계를 필요로 한다. 부적당한 구현은 열악한 성능을 쉽게 산출할 수 있다.

공표자 표명의 가장 일반적인 사용은 지정된 기업 엔티티에 관한 모든 완성된 공표자 표명을 검색하고 있는 find_relateBusiness API이므로, 각각의 표명을 그것이 참조하는 기업 엔티티 아래에 배치하는 것이 좋은 설계이다.

표명의 상태를 계산하고 그것을 표명 객체에 저장함으로써 검색을 완성된 공표자 표명으로 제한하는 것이 가능하다. 이것은 반환 결과들이 제거되어야 할 거짓 참조들을 포함하지 않게 된다는 것을 의미한다.

관계 객체를 보조 클래스로서 저장하는 것은 검색이 원하지 않는 관계를 가진 임의의 표명을 제거할 수 있게 해준다. 관계가 자식 객체로서 저장되는 경우, 관계 및 표명 완성 상태 양자에 어드레스하는 단일 검색을 기록하는 것이 불가능해진다.

존재할 경우 명명(naming)을 위해 UDDI 키들이 사용될 수 있다. UDDI는 중요한 객체 클래스들의 대부분에 대한 키들을 정의하며, 이들 키는 고유한 것을 보증하는 것으로 지정된다. 이것은 키들이 객체들에 대한 명명 속성들로서 사용될 수 있다는 것을 의미한다. UDDI 키를 명명 속성으로 사용하는 것은, 예를 들어 디폴트 이름이 기업 엔티티에 대한 명명 속성으로 사용되는 경우에 요구되는 명명 불일치의 해결을 시도할 필요가 없다는 것을 의미한다.

존재하지 않을 경우 명명을 위해 키들이 제공될 수 있다. 즉, 모든 UDDI 객체들이 정의된 키를 갖지는 않는다. 일례는 공표자 표명들이다. 이들에 대해, 본 시스템은 UDDI 정의 키들에 사용되는 것과 동일한 알고리즘을 이용하여 키를 제공한다. 이러한 아이디어의 재이용은 다른 객체들에 대해 기록된 코드 및 구조가 재사용될 수 있음을 의미한다.

일련의 UDDI 객체들이 다른 객체의 자식들이고, 자식들의 순서가 중요한 경우(예를 들어 어드레스 라인들), 자식 객체들에 할당되는 키들은 값이 단조롭게 증가하도록 배열되어, 키들에 대한 분류가 원하는 순서를 산출할 수 있게 한다. 이것은 원하는 순서를 보장하는 프로세스를 간단하게 한다.

실제의 경우, 키들은 리틀 엔디안(little-endian) 방식으로 변하는 것이 바람직하다. 즉, 키의 가장 좌측 바이트가 가장 빠르게 변하는데, 이것이 데이타 스토어로서 사용되고 있는 X.500 디렉토리에서 최상의 인덱싱 성능을 산출하기 때문이다.

UDDI 표준은 주요 객체 타입들의 일부 내의 하위 구조들의 수를 정의한다. 많은 경우에, 이들 하위 구조는 선택적이며, 반복될 수 있다(이들은 동일 객체 내에 0, 1 또는 2번 이상 발생할 수 있다). 간단한 예는 스트링(이름) 및 언어 식별자를 포함하는 이름 하위 구조이다. X.500 스키마 정의는 구조화된 속성들의 사용을 지원하지 않으며, 따라서 즉시 명료한 하위 구조들의 맵핑이 존재하지 않는다. 이러한 하위 구조들이 X.500 스키마에서 구현될 수 있는 몇가지 방법이 있다.

하나의 방법은 다양한 요소들을 분할하기 위한 소정 유형의 분리기를 사용하여 하위 구조의 컴포넌트들을 단일 속성 내에 연결하는 것이다. 이것은 최적의 설계 선택이 아닐 수 있는데, 이는 이 방법이 컴포넌트들을 개별적으로 인덱스 또는 검색하는 능력을 잃고, 데이타 처리에 프로세싱 복잡성을 추가하기 때문이다.

본 시스템에서, 하위 구조를 표현하는 데 이용되는 특정 설계는 성능 및 관리성을 최대화하도록 선택된다. 개시된 설계는 디렉토리에 하위 구조들을 표현하는 다양한 기술들 중 하나 이상을 이용할 수 있다. 이러한 기술들은 3가지 카테고리로 요약될 수 있다.

한가지 기술은 하위 구조들의 대부분이 자식 객체로서 처리될 수 있는 기술이다. 이름들은 훌륭한 예이며, 기업 엔티티 이름들은 기업 엔티티의 자식으로서 저장된다. 또 하나의 예는 기술(description)이며, 여기서 개별적인 기업 기술 객체는 기업 엔티티 객체의 자식이다. 도 15는 이러한 본 발명의 양태에 대한 일 실시예를 나타내며, 후술된다.

또 하나의 기술은 평면화(flattening)/병합(merging)이다. 많아야 한 개의 다른 객체에 대한 관계가 있을 수 있는 경우, 속성들은 단일 객체 내에 결합될 수 있다. 이 경우, 계층 구조는 평면화된다고 말하는데, 이는 2개의 객체가 하나의 객체로 결합되었기 때문이다. 새로운 객체는 병합되었다고 말하는데, 이는 새로운 객체가 결합 객체들로부터의 속성들의 결합을 포함하기 때문이다. 바람직하게, 관계 객체의 내용들은 부모 객체로 승진한다.

예를 들어, 도 16은 UDDI 관계도의 개략적인 표현이다. 도 17은 디렉토리 계층 구조가 UDDI 객체들의 평면화에 의해 형성된 디렉토리 계층 구조도를 개략적으로 도시하고 있다.

예를 들어, 도 16은 객체(163)의 관계 객체(162)를 가진 객체(161)를 나타낸다.

일 대 일 관계가 존재하는 본 발명의 일 실시예에 따르면, '자식'은 승진될 수 있다. 즉, 계층 구조의 그 부분은 무너지거나 평면화되고 객체들은 병합될 수 있다. 그 결과가 도 17에 개략적으로 도시된다. 부모 객체(171)는 내용들 A1, A2, An을 가지며, 내용들 B1, B2, Bn, C1, C2 및 Cn을 가진 하나 이상의 자식, 자식 객체 9n을 갖는다.

또 하나의 기술은 분할이다. 예를 들어, 하나의 특정 사례(OverviewDoc 하위 구조)에서, 하위 구조는 비반복 요소 및 반복 요소를 포함한다. 비반복 요소(OverviewURL)는 부모로 이동할 수 있는 반면, 반복 요소는 자식 객체로 될 수 있다.

본 발명의 다른 양태는 관리이다. T모델의 삭제는 T모델을 find_TModel에게 숨기지만, 저장소로부터 제거하지는 못한다. 따라서, 정확한 T모델의 처리를 구현하기 위하여, 숨겨진 플래그가 구현될 수 있다. 이러한 플래그의 존재는 T모델(또는 사용자 객체)이 숨겨져 있다는 것을 나타낸다. 이 플래그의 부재는 T모델이 숨겨져 있지 않다는 것을 나타낸다. 이것은 거의 대부분의 T모델에 대한 사례이며, 따라서 이러한 방법은 효율적이다. 숨겨지지 않은 객체들에는 공간이 존재하지 않으며, 인덱싱도 이용되지 않는다. 디렉토리는 숨겨진 속성을 가진 엔트리들만을 인덱스한다. 이것은 또한 숨겨지지 않은 T모델에 대한 검색이 빠르고 효율적이라는 것을 의미한다.

데이타 저장소로서 사용되는 X.500 디렉토리는 공백 값들을 저장하지 않는 설계를 장려한다. 예를 들어, 객체에 없는 (선택적인) 값은 디렉토리에 저장되지 않는다. 이것은 저장 공간을 효율적으로 이용할 수 있게 하며, 검색을 보다 강력하게 한다. 속성에 대한 임의의 검색은 그 속성에 대한 데이타를 가진 객체들을 고려하는 것만을 요구한다.

본 시스템의 데이타 계층 구조는 UDDI 표준의 의도와 잘 맞는다. UDDI 객체를 삭제하기 위한 요구가 도달한 때, 요구는 디렉토리 내의 서브 트리의 삭제로 직접 맵핑된다. 예를 들어, 서비스의 삭제는 그의 이름 및 설명, 및 그의 결합 템플릿들 모두의 삭제를 포함한다. 이들 모두는 디렉토리 내의 서비스 엔트리의 자식이다. 따라서, 본 시스템은 서비스 엔트리로부터 아래로 서브 트리를 삭제한다. 이것은 쉽게 구현되며 효율적이다.

도메인은 계층적 서브 트리의 토대를 표현하는 이름이다. X.500 용어에서 도메인은 컨텍스트 접두사로서 알려져 있다. LDAP 용어에서 도메인은 접미사로서 알려져 있다. UDDI 저장소들이 주어질 때, 도메인 이름은 저장소에서 데이타의 진정한 분산의 이용(X.500 의미에서)을 허용한다. UDDI 표준은 복제만을 지원한다. 도메인 노드들을 갖춤으로써 본 시스템은 애플리케이션에 투명하게 디렉토리 분산 설비를 이용할 수 있다.

예를 들어, 기업이 UDDI를 내부적으로 전개하지만 2개의 개발 사이트를 갖고 있다고 가정하자. 이러한 설비를 이용하여, 기업은 각각의 사이트에서 UDDI 서버를 전개할 수 있으며, 분산은 각각의 사이트가 양 레지스트리 상에 공표된 항목들을 투명하게 볼 수 있게 한다.

이것의 이점은 '무료로' 분산을 허용한다는 점이다. 예를 들어, UDDI 서버는 어떠한 추가 작업도 할 필요가 없으며, 디렉토리 시스템은 정보의 섬들을 효과적으로 함께 연결한다.

UDDI 표준에서 그 어떤 것도 사용자 정보가 어떻게 저장되는지를 구술하지 않는다. 사용자 객체를 생성함으로써, 사용자에 관한 모든 정보는 단일 객체에 저장될 수 있으며, 이 객체는 사용자가 공표하는 객체들의 모두를 유지하는 서브 트리의 루트로서 사용될 수 있다. 이것은 보안의 정의를 보다 간단하게 만든다. 예를 들어, 고려 중인 객체(기업, 서비스 또는 심지어 T모델)가 사용자의 사용자 객체 아래에 있는 경우, 사용자는 그 객체를 제어한다.

UDDI는 반복 요소들을 포함하는 객체들을 정의한다. 성능, 검색성 및 관리성과 같은 이익을 위해, 이러한 반복 요소들은 자식 객체로서 표현될 수 있다.

반복 구조 데이타를 자식 객체로서 저장하는 것은 디렉토리에서 데이타의 표현을 효율적으로 허용하며, 각각의 필드는 검색을 위해 개별적으로 이용 가능하다(그리고 인덱스된다).

예를 들어, 기업 엔티티 이름들은 기업 엔티티 객체의 자식들로서 저장될 수 있다. 또 하나의 예는 기업 엔티티 객체들 아래의 자식들로서 저장될 수 있는 기업 설명이다.

이러한 타입의 시스템의 이점은 이름의 검색(일반적인 UDDI 검색)을 허용하며, 엔트리의 DN이 그 이름이 속하는 객체의 DN을 제공한다는 점이다.

UDDI는 중복 '컨테이너' 노드들(속성들이 아니라 자식 하위 구조들을 포함하는 UDDI 구조들)을 정의한다. 이들은 질의의 결과들로부터 비교적 저비용으로 구축될 수 있으므로 제거될 수 있다. 몇몇 경우에, 속성들이 자식 노드에서 그의 부모로 승진되어, 디렉토리 표현으로부터 이제 중복하는 자식 노드를 제거할 수 있다.

예를 들어, tModelInstanceDetails는 어떠한 속성도 포함하지 않음에 따라 디렉토리 스키마에 표현되지 않는다. instanceDetails는 그의 자식 overviewDoc의 속성들이 있었으므로 그의 속성들이 tModelInstanceInfo 부모로 승진됨에 따라 디렉토리 스키마에 표현되지 않는다. 카테고리 및 식별자 가방들(bags)은 디렉토리에 표현되지 않으며, 그들의 내용들은 가방 소유자의 자식들이 된다.

이것의 이점은 디렉토리 내의 엔트리의 수를 줄인다는 점이다. 특히, 이것은 DIT의 깊이를 최소화하여 성능을 향상시킬 수 있다.

도 12는 본 발명의 일 실시예에 따른 계층 구조를 개략적으로 나타낸다. 하나 이상의 도메인 또는 접두사(121)가 제공된다. 각각의 도메인(121) 아래에 하나 이상의 사용자(122)가 있을 수 있다. 각각의 사용자(122) 아래에는 하나 이상의 T모델(123) 및 하나 이상의 기업 엔티티(BE; 124)가 제공될 수 있다. 각각의 기업 엔티티(124) 아래에는 하나 이상의 공표자 표명(PA; 125), 하나 이상의 기업 서비스(BS; 126) 및 하나 이상의 서비스 투영(SP; 127)이 제공될 수 있다. 각각의 기업 서비스(BS; 126) 아래에는 하나 이상의 결합 템플릿(BT; 128)이 제공될 수 있다. 특정 구현에 의해 필요할 때 별명들이 배치될 수 있다. 예를 들어, 서비스 투영 객체(들)(도 12에 삼각형으로 도시됨)이 기업 엔티티 객체(들)로부터 별명으로서 생성될 수 있다.

도 12에 나타난 바와 같은 스키마 설계의 이점은 본 명세서 전체를 읽음으로써 명백해질 것이다.

공표자 표명들은 이들이 참조하는 기업 엔티티들의 아래에 배치되는데, 이는 이들이 기업 키를 지정하고 공표자 표명을 통해 그 키와 관련된 모든 기업을 검색하는 find_RelatedBusiness 호출과 관련하여 가장 빈번하게 사용되기 때문이다. 본 시스템은 지정된 기업을 찾은 후 그 아래의 (완전한) 모든 공표자 표명들을 읽는다. 이것은 관련 표명을 찾는 빠르고 효율적인 방법이다.

이것의 이점은 빠르고 효율적인 검색을 가능하게 한다는 점이다. 이것은 또한 용이한 데이타 완전성의 유지를 가능하게 한다. 예를 들어, 기업이 삭제될 때, 어떠한 공표자 표명들도 자동으로 삭제된다.

T모델들은 이들을 공표한 사용자에 의해 변경(또는 회수/은닉)될 수 있다. 사용자를 표현하는 엔트리 아래에 이들을 배치하는 것은 보안을 간단하게 만든다. 예를 들어, T모델은 사용자 엔트리 아래의 서브 트리에 있는 경우에 수정될 수 있다. 그렇지 않은 경우에는 수정될 수 없다.

보다 상세하게는, 변경을 행하려고 하는 사용자의 DN이 T모델의 DN의 접두사와 일치하는 경우, 엔트리는 그 사용자에 의해 수정될 수 있으며, 그렇지 않은 경우에는 수정될 수 없다. 디렉토리는 이러한 결정을 만드는 데 사용되거나(DN이 존재하지 않는 경우에는 예외를 명명), UDDI 서버가 그것을 행할 수 있다.

객체가 저장소로부터 삭제될 때, 그 객체와 관련된 정보도 삭제될 수 있다. 이것은 본 스키마의 실시예들에 따라 사용되는 계층적 설계에 의해 매우 간단해진다. 객체가 삭제될 때, 이 객체가 루트인 전체 서브 트리는 삭제될 수 있으며, 이 프로세스는 모든(그리고 일반적으로 단지) 관련 정보를 삭제할 수 있다. 서브 트래의 삭제는 아래에서 위로 수행될 수 있다. 각각의 엔트리는 그의 모든 자식들이 삭제될 때에만 삭제될 수 있다. 이것은 모든 자식들을 DN 역순으로 리스트화함으로써 관리된다. 이것은 자식들의 부모 이전에 그 자식들이 삭제되는 것을 보증한다.

이것의 이점은 분류된 리스트 방법이 보다 복잡한 회귀의 사용에 대한 대안이라는 점이다. 또한, 이것은 비교적 간단하고 메모리 효율적이다. 서브 트리 내의 모든 엔트리가 DN에 의해 분류되고, 삭제가 역순으로 실행될 때, 이것은 모든 자식들이 그들의 부모 이전에 삭제되는 것을 보증한다.

예를 들어, 기업 서비스가 삭제될 때, 시스템은 그와 관련된 모든 결합 템플릿, T모델 인스턴스 정보 및 다양한 관련 카테고리 정보를 삭제한다. 이들 모두는 기업 서비스가 루트인 서브 트리를 삭제함으로써 삭제될 수 있다.

본 스키마의 설계에 사용되는 계층 구조로 인해, 객체의 DN은 객체에 대한 소유 및 제어의 체인을 나타낸다. 추론도 명명 속성들의 주의 깊은 선택에 의존한다는 점에 유의한다.

이것의 이점은 정보를 모으는 데 사용되는 검색 또는 판독의 수를 감소시킬 수 있다는 점이다. 예를 들어, 자식 객체들(이름 등)인 검색 결과에서, 각 엔트리의 DN은 부모(예를 들어 기업 엔티티) 및 소유 계정을 나타낸다.

예를 들어, 기업 서비스의 DN은 그것이 속하는 기업 및 그것을 제어하는 사용자를 나타낸다.

디렉토리는 어떠한 결과의 순서화도 보증하지 않는다. 복잡한 결과(기업 엔티티 및 그의 기업 서비스들, 이들의 적당한 이름들 및 설명들 등)를 처리할 때, 출력의 구축은 검색의 결과들을 취하고 이들을 DN에 의해 분류함으로써 간략화될 수 있다. 이것은 결과들을 체계화하여 결과들의 구축이 비교적 간단해지도록 한다. 각 객체는 그의 자식들 이전에 구축되어, 자식들을 그의 부모 아래에 배치하는 것을 용이하게 하며, 따라서 기업에 대한 결과가 그의 서비스들 이전에 체계화된다. 객체의 모든 자식들은 동일 타입의 다음 객체 이전에 나타나며, 하나의 기업에 대한 모든 서비스들이 다음 기업 이전에 나타난다. 이것은 또한 동일한 사항이 각 레벨에서 적용되므로 간단한 회귀 구축을 가능하게 한다.

이것의 이점은 UDDI 구조를 구축하는 데 필요한 로우 엔트리들(raw entries)의 리스트를 통한 전달의 수를 최소화한다는 점이다.

예를 들어, 분류 후, 기업 A에 대한 결과에는 그의 제1 서비스 AA에 대한 결과, 그 서비스의 이름이 이어지고, 그 다음 A의 제2 서비스 AB 및 그의 이름이, 그 다음에 제2 기업 B가 이어진다.

검색은 또한 자식들 상에서 수행될 수 있다. 예를 들어, 빈번한 검색 요구는 "그의 자식들 중 하나(또는 그 이상)를 발견함으로써 x를 찾는 것"일 수 있다. 기업이 검색에 의해 발견될 수 있는 방법들 중 하나는 예를 들어 결합 템플릿에서 발생하는 T모델을 지정하는 것이다. 즉, 질의는 "이러한 T모델을 참조하는 결합 템플릿을 가진 서비스를 가진 모든 기업을 찾는 것"이다. 이러한 질의는 후손 객체의 DN을 찾고 원하지 않은 레벨들을 삭제하여 기업 엔티티의 DN을 산출함으로써 행해질 수 있다. 이롭게도, 이것은 또한 사본을 제거한다. 이러한 검색 방법은 본 발명의 실시예들의 계층 구조에 부분적으로 기인하여 실현된다.

보증된 고유 키의 사용은 문제를 간단하게 한다. 전체 저장소는 단일 키에 대해 검색될 수 있으며, 고유성은 결과가 없거나(그 키가 없는 경우), 하나의 결과가 존재하는(그 키가 있는 경우) 것을 보장한다. 검색을 부모의 범위 내로 제한하는 것에 대해 조심할 필요가 없다. 이것은 최적 조건으로 데이타베이스 인덱스를 사용할 수 있으므로 디렉토리로부터 향상된 성능을 산출한다.

이것의 이점은 가장 빠른 타입의 디렉토리 질의를 사용한다는 점이다. 또 하나의 이점은 주어진 객체가 다른 객체로부터 참조되는 경우에 보증된 고유 이름들이 중요할 수 있다는 점이다.

대부분의 인덱싱 시스템들의 특성은 이들이 데이타 종속적이라는 점이다. 데이타가 "리틀 엔디안"(가장 좌측 부분이 가장 빠르게 변한다)인 경우, 그 데이타는 분산되는 경향이 있고, 따라서 인덱스들은 최대 성능을 제공할 수 있다. 역으로, 데이타가 반복적인 경우, 인덱스들은 그렇게 효과적이지 않을 수 있다. "리틀 엔디안" 품질을 보이는 UUID(Universally Unique Identifier) 알고리즘이 이용될 수 있다. 이것의 이점은 디렉토리 성능을 최대화한다는 점이다.

키는 도출된 객체에 추가될 수 있다. 반복 데이타 요소가 자식 객체로 되는 경우, 그의 DN의 마지막 아크를 형성하는 명명 속성을 추가할 필요가 있다. 디렉토리에서, 명명 속성은 그의 형제들과 다른데, 이는 동일 부모의 두 자식이 동일한 이름을 가질 수 없기 때문이다.

두 종류의 키가 사용될 수 있다. 순서를 요구하지 않는 자식 객체들에 대해서는 UUID가 사용되는데, 이는 이들이 고유한 것으로 보증되기 때문이다. 순서가 중요한 경우에는 순서를 보증하기 위하여 단조 증가 특성을 가진 키들이 사용된다.

UDDI 표준에서, 기업 엔티티는 2 종류의 서비스, 즉 기업 엔티티가 제어하는 서비스(자식 객체들에 의해 저장소에 표현됨), 및 다른 기업 엔티티에 의해 제공됨에도 불구하고 기업 엔티티가 인터페이스를 제공하는 서비스를 제공할 수 있다. 후자는 별명들에 의해 개시된 UDDI 저장소에 표현된다. 별명은 올바른 기능들은 정확하게 제공한다. 예를 들어, 오리지날 객체(서비스)가 그의 소유자에 의해 소정의 방식으로 변경되는 경우(아마도 다른 결합 템플릿이 추가됨), 이 객체는 별명 "변경"을 통해서도 참조된다. 더욱이, 서비스에 대한 기업 엔티티 아래의 임의의 검색은 실제 서비스 및 별명 서비스 양자를 산출한다.

예를 들어, 별명들은 서비스 투영을 위해 사용될 수 있는데, 기업은 다른 기업 아래에 정의되는 서비스를 지시할 수 있다.

이것의 이점은 별명의 이용이 자동으로 제공되는 "대체 이름"을 기본적으로 수반하는 기능을 허용한다는 점이다. 더욱이, 디렉토리가 별명 완전성을 지원하는 경우, 게다가 오리지날 서비스가 삭제되는 경우,임의의 투영들이 자동 제거된다.

UDDI 표준에서는 다른 객체를 직접 참조하는 것이 아니라 T모델 인스턴스 정보의 경우에서와 같이 중간 단계를 거치거나 공표자 표명에서 기업 엔티티들을 참조하는 다수의 장소들이 존재한다. 이 경우, 별명은 코드를 복잡하게 한다. 따라서, 그 대신 본 시스템은 객체에 대한 참조를 이용할 수 있다. 일 실시예에 따라 본 시스템은 모든 객체가 고유 키를 갖는 것을 보증하므로, 그 키는 참조로서 정확하게 거동하는데, 이는 종종 "외부(foreign)" 키로 알려져 있다.

보조 객체 클래스를 이용하여 속성 그룹화가 수행될 수 있다. 공표자 표명들의 처리에 있어서, 공표자 표명을 고유하게 식별하는 3개의 속성, 즉 2개의 기업 엔티티 키 및 이들 사이의 관계를 이용하여 공표자 표명을 찾는 능력이 필요하다. 그러나, 관계는 그 자체가 3개의 다른 속성들, 즉 T모델 키, 키 이름 및 키 값인 키를 갖춘 참조로서 지정된다. 하나의 방법은 이러한 관계를 공표자 표명의 자식 객체로서 저장하는 것이다. 그러나, 이것은 특정 공표자 표명에 대한 가장 효율적인 검색을 허용할 수 없다. 관계 키를 갖춘 참조를 공표자 표명 엔트리에 대한 보조 클래스로 간주함으로써 단일 검색에서 모두 5개의 속성을 검색하며, 따라서 필요한 공표자 표명 객체들을 정확하게 찾는 것이 가능하게 된다.

이러한 스키마에 대한 하나의 설계는 정상적인 객체 지향 설계 기술을 이용하여, 예를 들어 동일한 속성 이름을 가진 모든 키를 갖춘 참조들을 산출할 수 있다. 그러나, 이러한 설계는 예를 들어 기업 엔티티 카테고리 키를 갖춘 참조를 분리하여 T모델 카테고리 키를 갖춘 참조와 혼동하는 것을 피하는 것을 보다 어렵고 비싸게 만들 수 있다. 이것은 또한 필터에 객체 클래스 용어를 포함시키는 것이 필요하게 만들 수 있으며, 이러한 용어는 약하다(저장소 내에서 매우 반복적이다).

예를 들어, 모든 상이한 종류의 키를 갖춘 참조에 상이한 객체 클래스 및 상이한 속성 이름들을 제공하는 것은 특정 속성 이름에 대한 임의의 검색이 객체 클래스를 반드시 포함한다는 것을 의미한다. 이것은 또한 디렉토리 서버가 원하는 특정 종류의 엔트리에 대해 그 안에 엔트리들만을 갖는 인덱스를 구출할 수 있다는 것을 의미한다. 이러한 인덱스는 보다 작으며, 결과적으로 더 빠를 것이다.

예를 들어, "euBusinessEntityName=Smith"와 같은 검색은 인덱스에서 euBusinessEntityName을 찾을 것이며, 따라서 euTModelName이라고 하는 속성에서 Smith를 포함하는 엔트리와 혼동될 수 없다.

UDDI 표준의 범위 밖의 도구들에 대한 호출이 있을 수도 있다. 이러한 도구들은 UDDI 표준에서 지정된 것들 외의 액세스 수단을 제공하는 것을 요구할 수 있다. 이러한 도구들을 허용하기 위하여, 본 발명은 단일 UDDI 개념을 나타내는 모든 객체 클래스들을 결합하는 추상 클래스를 정의한다. 이것은 예를 들어 모든 이름들 또는 모든 키를 갖춘 참조들을 볼 수 있는 검색의 정의를 가능하게 한다.

예를 들어, euBusinessEntityName 및 euTModelName을 포함하는 모든 이름 타입 객체 클래스들의 수퍼 클래스인 추상 클래스 euName이 존재한다.

UDDI 표준은 예를 들어 사례에 민감한 방식 및 사례에 민감하지 않은 방식으로 이름들을 검색하는 것이 가능하도록 지정한다. 이것은 엔트리들을 사례에 민감하지 않게 인덱스하여 검색하고 사례에 민감하게 검사함으로써 처리될 수 있으나, 이러한 방법은 성능을 저하시킨다. 이들의 경우에, 동일 데이타를 포함하지만 다르게 인덱스되는 쉐도우 필드를 정의하는 것이 바람직하다. 마찬가지로, 언어의 변화에 대해 예를 들어 발음 구별 부호와 같은 쉐도우 속성들이 사용될 수 있다.

예를 들어, euBusinessEntityName 객체 클래스는 각 이름의 2개의 사본을 포함한다. 제1 버젼은 사례에 민감하지 않게 인덱스되지만, 제2 버젼은 사례에 민감하게 인덱스된다. 이것은 어떠한 거동이 요구되는지에 관계 없이 최적으로 수행하는 검색 필터의 구축을 가능하게 한다.

이 저장소 내의 모든 속성(객체 클래스 제외)은 단일 값을 가질 수 있다. 이것은 디렉토리가 보다 효율적인 인덱스를 구축하여 보다 양호한 검색 성능을 제공하는 것을 가능하게 한다.

이것은 또한 거짓의 긍정적인 검색 결과의 가능성을 제거한다. 예를 들어, "Fr"로 시작하여 "nk"로 끝나는 이름들을 찾는 검색을 고려하자. 이것이 "Frank"와 같은 이름을 가진 (유효) 엔트리를 산출할 것을 기대할 수 있다. 그러나, 이름이 다가 속성인 경우, "Fred" 및 "Tink"와 같은 2개의 이름을 가진 무효 엔트리를 얻을 수도 있는데, 이것은 이 하나의 엔트리가 지정된 양쪽 기준에 일치하기 때문이다. 각각이 엔트리의 자식 객체인 단일 값 이름들을 사용함으로써, "Fred"와 "Tink"의 거짓 일치가 제거된다.

조작 속성들은 UDDI 애플리케이션에 의해 관리되지만 사용자에게 보이지 않는 특수 속성들이다.

UDDI 데이타의 저장에 있어서, 사용중인 T모델을 회수된 것들과 구별하는 방법을 갖는 것이 가능해야 한다. T모델이 삭제될 때, T모델은 여전히 많은 엔트리들에 의해 사용될 수도 있으며, 따라서 T모델은 진정으로 삭제될 수 없다. 대신에, T모델은 숨겨지는데, 이는 T모델이 find_TModel 호출의 결과들의 일부로서 반환되지 않지만 get_TModelDetail 호출을 통해 여전히 질의될 수 있다는 것을 의미한다. 이것은 숨겨진 T모델들에 추가되는 euHidden이라고 하는 속성을 이용하여 구현된다. euHidden 속성을 포함하는 임의의 엔트리를 제거하는 검색 단계를 T모델을 검색하는 임의의 필터에 추가하는 것이 이롭고 효율적일 수 있다.

디렉토리 구현에 있어서, 현저하게 하나의 값인 속성을 갖는 것은 일반적으로 매우 비효율적인 것으로 간주된다. 예를 들어, 엔트리들의 99%에 대해 거짓으로 설정된 숨겨진 속성을 갖는 것은 열악한 성능을 산출하게 되며, 인덱스는 훨씬 더 사용 불가능하게 될 것이다.

보다 더 효율적인 것으로 간주되는 것은 엔트리들의 대부분을 숨겨진 속성 없이 저장하고 숨겨질 엔트리들에만 속성을 추가하는 것이다. 이것은 모든 "거짓" 값을 유지하기 위한 저장 공간을 필요로 하지 않는 추가적인 이익을 갖는다. 이제, 숨겨지지 않은 모든 T모델을 발견하기 위한 필터는 "(!(euTModel=*))"이 되는데, 이것은 존재 테스트의 부정이며, 존재 테스트는 속성들이 엔트리들 중 작은 부분 상에만 존재할 때 특히 빠르다.

이제, 디렉토리와 관련된 구현 및 UDDI 표준의 문제를 해결하기 위한 본 발명의 일 실시예가 설명된다. X.500 스키마에 대한 다수의 요소들이 존재한다. 이들 요소는 속성 정의, 객체 클래스 정의 및 이름 결합 정의를 포함한다. 속성 정의는 단일 데이타 요소를 지정하고, 이 요소에 고유 식별자(OID), 이름 및 데이타 타입을 제공한다. 객체 클래스 정의는 전체로서 조작되는 속성들의 집합을 지정한다. 이것은 고유 식별자(OID), 이름 및 속성 리스트를 제공하는데, 속성들은 요구되거나 선택적일 수 있다. 이름 결합은 가능한 계층 구조의 일부를 지정한다. 이름 결합은 다른 객체 클래스 아래에 저장될 수 있는 하나의 객체 클래스를 지정하며, 이러한 관계에서 자식 객체를 명명하는 자식의 속성(또는 속성들)을 지정한다.

추가적인 설계 요건을 부과하는 다수의 발견 한정사가 존재한다. 하나의 발견 한정사는 사례에 민감한 방식 및 사례에 민감하지 않은 방식으로 텍스트 데이타를 효율적으로 검색하는 능력을 제공하는 사례 민감도이다. 본 발명의 일 실시예에 따르면, 사례 민감도는 상이하게 인덱스된 객체들에 추가 필드를 제공함으로써 해결될 수 있다.

이 실시예에 따르면, 텍스트 데이타는 타입 caseExactString의 속성에, 그리고 타입 caseIgnoreString의 속성에 두번 저장된다. 그 후, 발견 한정사는 어느 필드가 검색되는지를 결정하며, 이에 따라 최대 성능을 얻게 된다.

예를 들어, 기업 엔티티가 McKenna's Iron Foundry Services"와 같은 이름을 가진 경우, 이 스트링은 사례에 민감하게 인덱스되는 필드에 한번, 사례에 민감하지 않게 인덱스되는 필드에 한번, 총 2번 저장되는데, 저장된 데이타는 동일하지만, 기반 디렉토리에 의해 생성되는 인덱스들은 상이하다.

또 하나의 문제는 서비스 투영을 효율적으로 구현하는 것이다. 본 발명의 일 실시예에 따르면, 이것은 X.500 별명 설비를 이용하여 해결될 수 있다. 서비스 투영을 처리할 수 있는 많은 방법이 존재한다. 본 발명의 이 실시예는 디렉토리 별명들에 의해 서비스 투영을 처리한다. 이것은 서비스 투영을 구현하는 데 특히 효율적인 방법이다. 이것은 투영의 기본 서비스와의 일관성을 보증하는데, 이는 기본 서비스가 별명을 통해 직접 액세스되기 때문이다. 이것은 또한 투영이 기본 서비스가 삭제되는 순간에 사라지는 것을 보증함으로써, 일관성을 보장한다.

예를 들어, Williams Accounting Services라고 하는 기업 엔티티가 General Ledger Cross-Check라고 하는 웹 서비스를 공표하고, Williams Auditing Services라고 하는 제2 기업 엔티티 아래에 동일 서비스를 제공하기를 원하는 경우, 이것은 제2 기업 엔티티 아래에 별명 엔트리를 배치함으로써 달성될 수 있다. Williams Auditing Services에 의해 제공되는 서비스들을 열거하는 조회자는 Williams Auditing Services에 의해 직접 제공되는 임의의 서비스들을 발견할 때만 General Ledger Cross-Check 서비스를 발견하게 된다.

또 하나의 문제는 키를 효율적으로 구현하는 것이다. 본 발명의 일 실시예에 따르면, 이것은 외부 키에 대한 UUID, 및 순서가 중요하지 않은 키를 이용하여 해결된다. 순서가 중요한 경우 순번이 사용될 수 있다. 키들은 스트링으로 표현되지만, 이들은 진정으로 텍스트 데이타는 아니다. 이들은 민감도 없이 사례 또는 발음 구별 부호와 비교된다.

외부에서 볼 수 있는 키들은 일련의 규칙들을 따른다. UDDI 사양의 버젼 2에 따르는 저장소를 구현할 때, 이들은 ISO-11578에 따라 UUDI를 유지한다. UDDI 사양의 버젼 3에 따라 저장소를 구현할 때, 이들은 그 사양 버젼에 정해져 있는 규칙들에 따라 키 스트링들을 유지한다.

요소들을 함께 연결하기 위해 내부적으로 사용되는 키들은 다른 일련의 규칙을 따른다는 점에 유의한다. 순서가 중요하지 않은 키들은 UUID를 사용한다. 순서가 중요한 경우, 순번이 사용된다.

예를 들어, Williams Auditing Services라고 하는 기업 엔티티에 대한 카테고리 가방의 요소를 나타내는 키를 갖춘 참조는 12345678-1234-1234-1234-1234567890ab의 키(UDDI v2)를 가진 T모델을 참조할 수 있다. 카테고리 가방에서 키를 갖춘 참조들의 순서는 중요하지 않지만, 키를 갖춘 참조는 키가 객체의 명명 속성으로서 기능할 것을 요구한다. 따라서, 이러한 객체에 대해 87654321-4321-4321-4321-ba0123456789와 같은 UDDI 키를 생성하고 이것을 이 객체에 대한 디렉토리 내의 명명 속성으로서 사용할 수 있다.

또 하나의 문제는 X.500 분산을 원하는 경우 데이타가 도메인 내에 체계화될 수 있다는 점이다. 이것은 본 발명의 일 실시예에 따라 사용자들 위에 저장소 계층을 생성하여 각각의 저장소가 상이한 서버 상에 배치될 수 있게 함으로써 해결된다.

UDDI 표준은 이름 공간이 분산되는 것을 허용하지 않는다. 이것은 다수의 UDDI 레지스트리들이 복제에 의해, 또는 분산된 이름 공간들을 관리하는 백엔드 데이타 스토어를 투명하게 구비함으로써 서로 협조할 수 있다는 것을 의미한다.

이름 공간의 분산은 명명 접두사를 가진 각각의 저장소에 의해 용이해질 수 있다. 이러한 접두사는 도메인을 정의하는 일련의 노드들이다. 이러한 노드들은 각각의 UDDI 레지스트리 위의 저장소 계층으로 간주될 수 있다. 이들 노드는 사용자 레벨 위에 배치된다.

도 11은 "도메인"(110)이라고 하는 노드의 예를 나타낸다. 도메인(110)은 디렉토리 접두사이며, 루트까지 하나 이상의 노드를 포함할 수 있다. 이 예는 도메인(110) 아래에 예를 들어 다수의 사용자들(112, 113, 114)의 배열을 도시하고 있다. 도메인(110) 아래에 배치된 사용자들의 수는 본 발명의 특정 구성 및/또는 용도에 따라 변할 수 있다. 또한, 본 발명의 특정 구성 및/또는 용도에 따라 다수의 도메인들이 배열될 수도 있다. 아래의 예에서, 이들은 저장소 객체들로 지칭되어, 이들이 개별적인 물리적 저장소들을 나타낸다는 것을 암시한다. 물론, 이것은 반드시 본 발명의 구성 및/또는 용도에 따르는 경우는 아닐 수 있다.

저장소 객체는 명명 속성을 요구하지만, 그것이 전부이다.

```
set object-class uddiObjectClass:400 =
{       # repository - may be used to break users into groups
        name = euRepository
        subclass-of top
                must-contain
                        euRepositoryName
};
```

분산은 복제의 대규모 대역폭 오버헤드 및 동기화 문제 없이 다수의 노드들에 의해 데이타가 공유되는 것을 허용하므로 대규모 디렉토리 전개에 있어서 중요한 개념이다.

일 실시예에서, 'eTrust' UDDI는 기반 eTrust 디렉토리 서버의 능력을 이용하여 분산을 지원하며, 이것이 잘 동작하도록 하기 위해 스키마가 적절히 구축되었고, 트리 계층 구조의 최상부에 가상 '도메인' 노드(들) 및 각 노드 서브 트리의 최상부에 고유 노드 식별자 또는 이름들이 허용된다(아래의 UDDI 스키마 설명을 참조).

더욱이, eTrust UDDI 서버는 구성을 통해 '분산 인식적'으로 될 수 있다. 2개의 개별 디렉토리 접두사, 즉 검색 및 판독을 위한 접두사와 엔트리 추가를 위한 접두사가 지정될 수 있다. 분산 서버를 전개하기 위하여, 기반 eTrust 디렉토리 서버 에이젼트들이 eTrust 디렉토리 관리 지침서에 따라 분산을 위해 구성된다. 각각의 개별 eTrust UDDI 노드는 고유 노드 이름을 갖도록 구성된다. 각각의 노드에 대한 검색/판독 접두사는 'World' 또는 'Corporation' 노드 이름으로 설정된다. 각각의 노드에 대한 Add 접두사는 그 노드의 고유 이름으로 설정된다.

이러한 방식으로, 각 노드는 그 자신의 디렉토리 저장소에 엔트리들을 추가하지만, X.500 디렉토리의 분산 기능을 통해 모든 노드의 엔트리들을 검색한다.

저장소 객체의 일례는

euRepositoryName=Melbourne

일 수 있다.

또 하나의 문제는 사용자에 대해 유지되는 데이타를 체계화하는 것이다. 이것은 데이타를 유지하는 사용자 객체를 생성함으로써 해결될 수 있다.

UDDI 사양에는 사용자 객체가 지정되어 있지 않지만, 본 발명의 일 실시예에 따라 사용자 객체가 이용될 수 있다. 예를 들어, 사용자 객체는 여러 가지 중에서 사용자 인증서에 대한 저장점, 및 공표에 대한 지지점일 수 있다.

도 10은 '사용자'(101)라고 하는 배열의 예를 나타낸다. 이 예는 사용자(101) 아래에 기업 엔티티 객체(102), 기업 서비스 객체(103) 및 결합 템플릿 객체(104)와 같은 다른 객체들의 배열을 도시하고 있다. 사용자(101) 아래에 배열된 기업 엔티티 객체의 수는 본 발명의 특정 구성 및/또는 용도에 따라 변할 수 있다. 또한, 본 발명의 특정 구성 및/또는 용도에 따라 다수의 사용자들이 배열될 수도 있다.

사용자 객체에 유지되는 데이타 요소들은 사용자 키(사용자 계정에 대한 고유 이름을 제공하기 위해 사용됨), 사용자 이름, 및 인증서(패스워드처럼 간단하거나 PKI 인증서처럼 복잡할 수 있음)를 포함한다. 또한, 이것은 인증된 이름(사용자 계정을 조작할 수 있도록 인증된 사람 또는 역할을 식별함)을 포함할 수도 있다. 이것은 또한 사용자에 의해 정의된 임의의 T모델을 분실하는 일 없이 사용자 계정의 삭제를 처리하는 데 사용되는 숨겨진 플래그를 포함할 수도 있다.

```
set object-class uddiObjectClass:401 =
{       # user account
        name = euUserAccount
        subclass-of top
        must-contain
                euUserKey,
                euUserName,
                euCredentials
        may-contain
                euAuthorizedName,
                euHidden
};
```

사용자 계정 객체의 일례는

euUserKey=23456789-2345-2345-2345-234567890abc

euUserName=Grace

euCredentials=Amazing76sQ

일 수 있다.

(이 예에서는 간단한 사용자 ID 및 패스워드 시스템이 구현된 것으로 가정한다.)

또 하나의 문제는 기업 엔티티(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적으로 표현하는 것이다. 이것은 본 발명의 일 실시예에 따라 고유 필드를 객체의 속성으로서 표현하고 요소들을 자식들로서 반복함으로써 해결된다.

기업 엔티티 객체는 UDDI 표준의 기본 컴포넌트이다. 그 내용은 표준에 의해 정의되지만, 그 요소들의 대부분은 X.500 스키마에 의해 지원되지 않는 복잡한 객체들을 반복하고 있다. 이러한 요소들은 계층적인 배열에 의해 표현된다.

기업 엔티티에 유일하게 필요한 요소는 기업 키이다. 선택적인 요소들은 인증된 이름, 오퍼레이터, 및 사용자 키(이것은 정상적인 사용자에 의해 공표되는 기업 엔티티에 존재한다)를 포함한다.

```
set object-class uddiObjectClass:402 =
{       # Business Entity - details of an entity which provides services
        name = euBusinessEntity
        subclass-of top
        must-contain
                euBusinessEntityKey
        may-contain
                euParentUserKey,
                euAuthorizedName,
        };
```

기업 엔티티의 가능한 자식 객체들은 이름(순서화를 위해 키를 갖춘 이름 스트링과 언어 코드를 포함하는 객체); 설명(순서화를 위해 키를 갖춘 설명 스트링과 언어 코드를 포함하는 객체); 컨택(후술되는 복합 객체); 발견 URL(키를 갖춘 URL 스트링과 사용자 타입을 포함하는 객체); 객체 클래스의 선택을 통해 카테고리 또는 식별자 정보로서 표시되는 키를 갖춘 참조들; 및 기업 서비스들(후술됨)이다.

기업 엔티티 객체의 일례는

euBusinessEntityKey=34567890-3456-3456-3456-34567890abcd

euParentUserKey=23456789-2345-2345-2345-23456789abc

일 수 있다.

기업 엔티티 객체의 명백한 내용의 대부분은 기업 엔티티 객체의 직계 자식들인 객체들에 실제로 저장된다는 점에 유의한다.

도 15는 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있다. 도 15에서, 다가 요소들인

For child 152

Language en

Name CA

For child 153

Language IN

Name CATS

는 기업 엔티티(151)의 자식들(152, 153)로서 표현된다. 자식들이 없을 수도 있고, 더 있을 수도 있다.

해결되어야 할 또 하나의 문제는 기업 서비스(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적인 방식으로 표현하는 것이다.

이것은 본 발명의 일 실시예에 따라 고유 필드들을 객체의 속성들로서 표현하고 반복 요소들을 자식들로서 표현함으로써 해결될 수 있다.

기업 서비스는 적어도 2가지 방법으로 구현될 수 있다. 첫 번째는 기업 서비스가 결합 템플릿에 의해 각각 표현되는 하나 이상의 액세스 라우트를 통해 이용 가능한 기업 엔티티에 의해 제공되는 단일 개념 서비스를 표현하는 것이다. 두 번째는 기업 서비스가 개별 서비스들로의 분할이 결합 템플릿 레벨에서 발생하는 서비스 그룹화 메카니즘인 것이다. 어느 경우에나, 데이타 필드는 UDDI 사양에서 정의된다.

기업 서비스의 요소들은 기업 및 서비스 키이다. 기업 키는 서비스를 소유하는 기업 엔티티를 지정한다. 기업 키는 반드시 기업 엔티티 아래에서 발견되는 것은 아니다. 단일 서비스는 서비스 투영에 의해 여러 기업 엔티티 아래에서 발견될 수 있다. 서비스 키는 UDDI 저장소를 통한 서비스의 고유 식별자이다. 이 두 키는 스트링으로 표현된다.

```
set object-class uddiObjectClass:403 =
{       # business
        name = euBusinessService
        subclass-of top
        must-contain
                euBusinessServiceKey,
                euParentBusinessKey
};
```

기업 서비스 객체의 선택적인 내용은 존재하지 않는다. 모든 다른 내용은 잠재적으로 반복하는 요소들로 이루어지며, 따라서 자식 객체로서 표현된다. 기업 서비스의 잠재적인 자식 객체들은 결합 템플릿(아래 참조); 이름(순서화를 위해 키를 갖춘 이름 스트링과 언어 코드를 포함하는 객체); 설명(순서화를 위해 키를 갖춘 설명 스트링과 언어 코드를 포함하는 객체); 및 카테고리 정보로서 표시되는 키를 갖춘 참조들이다.

예를 들어, 기업 서비스 객체는

euBusinessServiceKey=4567890a-4567-4567-4567-4567890abcde

euParentBusinessKey=34567890-3456-3456-3456-34567890abcd

이다.

기업 서비스 객체의 명백한 내용의 대부분은 기업 서비스 객체의 직계 자식인 객체들에 실제로 저장된다는 점에 유의한다.

도 15가 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있지만, 이것은 기업 서비스에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 동일하게 나타내고 있는 것이다. 도 15의 기업 엔티티(151)는 기업 서비스에 동일하게 적용될 수 있으며, 기업 서비스의 다가 요소들은 기업 서비스(151)의 자식들(152, 153)로서 표현된다. 자식이 없을 수도 있고 더 있을 수도 있다.

또 하나의 문제는 결합 템플릿(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적인 방식으로 표현하는 것이다. 이것은 본 발명의 일 실시예에 따라 고유 필드를 객체의 속성으로 표현하고 반복 요소들을 자식으로서 표현함으로써 해결된다.

결합 템플릿은 특정 서비스가 액세스될 수 있는 방법을 표현한다. 결합 템플릿의 유일하게 요구되는 요소들은 그의 키, 및 그것이 적용되는 서비스의 키이다. 선택적인 요소들은 액세스 점 또는 호스팅 재지향자(hosting redirector)를 포함할 수 있다(객체는 이들 중 정확하게 하나를 가져야 한다). 액세스 점이 존재하는 경우, 액세스 점 타입도 존재해야 한다.

```
set object-class uddiObjectClass:404 =
{       # binding template
        name = euBindingTemplate
        subclass-of top
        must-contain
                euBindingTemplateKey
        may-contain
                euParentServiceKey,
                euHostingRedirector,
                euAccessPoint,
                euAccessPointType
};
```

결합 템플릿의 가능한 자식 객체들은 T모델 인스턴스 정보(아래 참조); 및 설명(순서화를 위해 키를 갖춘 설명 스트링과 언어 코드를 포함하는 객체)이다.

결합 템플릿의 일례는

euBindingTemplateKey=567890ab-5678-5678-5678-567890abcdef

euParentServiceKey=4567890a-4567-4567-4567-4567890abcde

euAccessPoint=http://www.rsps.com.au/wsep

euAccessPointType=http

일 수 있다.

다시, 도 15는 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있지만, 이것은 결합 템플릿에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 동일하게 나타내고 있는 것이다. 도 15의 기업 엔티티(151)는 결합 템플릿에 동일하게 적용할 수 있으며, 결합 템플릿의 다가 요소들은 결합 템플릿(151)의 자식들(152, 153)로서 표현된다. 자식이 없을 수도 있고 더 있을 수도 있다.

또 하나의 문제는 T모델(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적인 방식으로 표현하는 것이다. 본 발명의 일 실시예에 따르면, 이것은 고유 필드를 객체의 속성으로 표현하고 반복 요소들을 자식들로서 표현함으로써 해결될 수 있다.

T모델은 아이디어를 표현한다. 이 아이디어는 예를 들어 검증될 수 있는 값들의 사양을 요구하는 카테고리화 시스템일 수 있다. 또는, 데이타 통신 프로토콜의 사양일 수 있다. T모델은 유연하고 강력한 개념이며, 복잡한 데이타를 정확하게 질의될 수 있는 방식으로 표현할 수 있는 UDDI의 능력에 중요하다.

T모델 객체의 유일하게 필요한 요소들은 T모델 키 및 이름이다. 이들은 스트링으로 표현된다.

T모델 객체의 선택적인 요소들은 인증된 이름, 개요 URL(개요 문서 객체의 일부), 사용자 키 및 숨겨진 플래그이다.

숨겨진 플래그는 T모델의 처리의 요소이다. 숨겨진 플래그는 deleteTModel 호출이 처리되는 방법이다. T모델이 삭제될 때, 숨겨진 플래그는 객체에 추가된다. 이것은 객체가 findTModel 호출로 반환되지 않지만 getTModel 호출에 액세스될 수 있다는 것을 의미한다.

```
set object-class uddiObjectClass:405 =
{       # tmodel - a reference to an idea.
        name = euTModel
        subclass-of top
        must-contain
                euTModelKey,
                euTModelName
        may-contain
                euAuthorizedName,
                euOperator,
                euOverviewURL,
                euParentUserKey,
                euHidden
};
```

가능한 자식 객체들은 설명(순서화를 위해 키를 갖춘 설명 스트링과 언어 코드를 포함하는 객체); 카테고리 또는 식별자 정보로서 표시되는 키를 갖춘 참조; 및 개요 문서 설명(순서화를 위해 키를 갖춘 설명 스트링과 언어 코드를 포함하는 객체)이다.

T모델의 일례는

euTModelKey=uuid:67890abc-6789-6789-6789-67890abcdef1

euTModelName=Corporate QA Policy

euOverviewURL=http://www.rsps.com.au/policy/qa.html

euParentUserKey=23456789-2345-2345-2345-234567890abc

일 수 있다.

다시, 도 15는 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있지만, 이것은 T모델에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 동일하게 나타내고 있다. 도 15의 기업 엔티티(151)는 T모델에 동일하게 적용될 수 있으며, T모델의 다가 요소들은 T모델(151)의 자식들(152, 153)로서 표현된다. 자식들이 없을 수도 더 있을 수도 있다.

또 하나의 문제는 공표자 표명(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적인 방식으로 표현하는 것이다.

본 발명의 일 실시예에 따르면, 이것은 고유 필드를 객체의 속성으로 표현하고 필요한 관계 키를 갖춘 참조에 대한 보조 클래스를 사용함으로써 해결될 수 있다.

공표자 표명은 2개의 기업 엔티티 사이의 관계를 표현하는 객체이다.

공표자 표명의 필요 요소들은 그의 키, 양쪽 기업들('to' and 'from' businesses) 기업 및 사용자 키, 상태 및 관계이다. 관계는 키를 갖춘 참조로서 지정되며, 공표자 표명 엔트리에 대한 보조 클래스로서 저장된다. 상태는 스트링으로 저장되지만, 완성 상태 객체로부터 그의 가능한 값들을 인출한다. 모든 키들은 스트링으로 표현된다.

```
set object-class uddiObjectClass:406.=
{       # publisher assertion - a relationship between two businesses
        name = euPublisherAssertion
        subclass-of   top
        must-contain
                euPublisherAssertionKey,
                euFromBusinessKey,
                euFromUserKey,
                euToBusinesKey,
                euToUserKey,
                euPublisherAssertionStatus
}
```

공표자 표명에는 선택적인 내용이 없으며, 자식 객체들도 존재하지 않는다.

공표자 표명의 일례는 다음과 같을 수 있다:

```
euPublisherAssertionKey = 7890abcd-7890-7890-7890-7890abcdef12
euFromBusinessKey = 34567890-3456-3456-3456-34567890abcd
euFromUserKey = 23456789-2345-2345-2345-234567890abc
euToBusinessKey = 09876543-6543-6543-6543-dcba09876543
euToUserKey = 98765432-5432-5432-5432-cba098765432
euPublisherAssertionStatus = status:complete
```

이 엔트리와 관련된 보조 클래스가 존재하며, 이것은 객체 클래스 euPublisherAssertionRelationshipKeyedReference이고, 명명된 2개의 기업 엔티티 사이에 표명되고 있는 관계를 지정한다는 점에 유의한다. 일례는 다음과 같을 수 있다:

```
euPublisherAssertionTModel=uuid:807A2C6A-EE22-470D-ADC7-
E0424A337C03
euPublisherAssertionKeyName = wholly-owned subsidiary
euPublisherAssertionKeyValue = parent-child
```

또 하나의 문제는 키를 갖춘 참조(UDDI 표준에서 설명되는 객체 클래스)에 관한 데이타를 효율적인 방식으로 표현하는 것이다. 이것은 키를 갖춘 참조들의 특정 집합, 예를 들어 기업 엔티티 상의 카테고리 가방을 효율적으로 검색할 수 있는 필요에 의해 보다 복잡해진다.

이것은 본 발명의 일 실시예에 따라 키를 갖춘 참조들을 표현하는 추상 기본 클래스를 생성하고 이것을 원하는 집합들 각각에 대해 서브클래스화함으로써 해결된다. 집합들은 디렉토리에서 표현을 갖지 않는다. 예를 들어, 집합들은 동일 객체의 자식들로서 존재하는 동일 서브클래스의 키를 갖춘 참조들의 그룹 이상으로는 존재하지 않는다. 예를 들어, 기업 엔티티의 카테고리 가방은 지정된 기업 엔티티의 자식들인 클래스 euBusinessEntityCategoryKeyedReference의 객체이다. 기업 엔티티 객체는 여러 개의 키를 갖춘 참조 객체들을 자식들로서 가질 수도 있으며, 이들의 객체 클래스들만이 어느 것이 카테고리 가방의 일부이고 어느 것이 식별자 가방의 일부인지를 분명하게 한다는 점에 유의한다.

키를 갖춘 참조들은 UDDI 데이타 모델 내의 여러 장소에서 사용된다. 이들은 T모델 키, 키 이름, 및 키 값을 포함한다. 키를 갖춘 참조들의 2가지 이용은 카테고리 가방 및 식별자 가방이다. 이 가방들은 키를 갖춘 참조들의 집합이며, 검색에 중요한 것이다. 이 가방들이 차별화되지 않은 키를 갖춘 참조들을 포함하는 객체들로 표현되는 경우, 효율적인 검색을 구현하는 것은 잠재적으로 아주 어렵게 된다. 이것은 키를 갖춘 참조들의 여러 서브클래스들이 구현되고 있는 이유이다. 기업 엔티티 상의 카테고리 가방은 클래스 euBusinessEntityCategoryKeyedReference의 하나 이상의 자식 객체에 의해 표현된다. 이것은 그들의 카테고리 가방에 지정된 키를 갖춘 참조들을 구비한 기업 엔티티들에 대한 효율적인 검색을 구현하는 것을 용이하게 해준다.

아래의 예는 추상 클래스, 및 전술한 바와 같이 도출된 클래스들 중 하나인 euBusinessEntityCategoryKeyedReference를 나타낸다. 키를 갖춘 참조에 대한 키는 추상 클래스로부터 상속되는 반면, T모델 키, 키 이름, 및 키 값은 도출된 클래스에서 모두 지정되며, 따라서 이들은 검색을 위해 다른 이름들을 가질 수 있다는 점에 유의한다.

```
set object-class uddiObjectClass:201 =
{       # abstract class as parent for all keyed references
        name = euKeyedReference
        subclass-of   top
        must-contain
                euKeyedReferenceKey
};

set object-class uddiObjectClass:301 =
{       # Business Entity category keyed reference - collection makes up the
category bag
        name = euBusinessEntityCategoryKeyedReference
        subclass-of euKeyedReference
        must-contain
                euBusinessEntityCategoryTModel,
                euBusinessEntityCategoryKeyName,
                euBusinessEntityCategoryKeyValue
};
```

컨택은 다양한 정보를 표현하는 복합 객체이다. 기업 엔티티와 같이, 컨텍은 자식 객체 클래스들의 이용을 필요로 하는 다양한 복합 반복 요소들을 유지한다.

직접적으로 컨텍 객체의 일부인 유일한 데이타 요소들은 키, 및 컨택에 표현하는 사람 또는 역할의 이름이다. 선택적인 사용 타입이 존재한다.

모든 다른 가능한 요소들은 컨택 객체의 자식들이다. 이들은 어드레스(각각 키, 사용 타입, 분류 코드 및 T모델 키를 구비한 어드레스 라인 객체들의 순서화된 리스트의 부모); 전화(전화 번호+사용 타입); 이메일(이메일 어드레스+사용 타입); 및 설명(설명 스트링+언어 코드)이다.

다시, 도 15는 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있지만, 이것은 컨택 객체에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 동일하게 나타내고 있다. 도 15의 기업 엔티티(151)는 컨택 객체에 동일하게 적용될 수 있으며, 컨택 객체의 다가 요소들은 컨택 객체(151)의 자식들(152, 153)로서 표현된다. 자식이 없을 수도 더 있을 수도 있다.

또 하나의 문제는 이름 및 설명(UDDI 표준에서 지정됨)을 효율적인 방식으로 표현하고 특정 타입의 이름 또는 설명에 대한 빠른 검색을 가능하게 하는 것이다.

본 발명의 일 실시예에 따르면, 시스템은 이름을 표현하기 위한 추상 기본 클래스, 및 설명을 표현하기 위한 또 하나의 추상 기본 클래스를 생성하고, 이들을 원하는 타입들 각각에 대해 서브클래스화한다. 특정 타입의 이름(예를 들어 기업 엔티티 이름)을 찾을 때 서브 클래스의 속성을 검색하고, 임의의 이름을 찾을 때에는 추상 클래스를 검색한다.

주요 객체들(기업 엔티티, 기업 서비스 등) 중 여러 객체는 다수의 이름 및 설명에 대한 옵션을 갖는다. 그 이유는 다양하다. 기업이 다수의 이름, 즉 아마도 하나의 정식 이름과 하나 이상의 통용 이름으로 알려지는 것을 일반적이다. 더욱이, 기업은 여러 언어의 여러 이름을 사용할 수 있다. 예를 들어 이름이 잘못 번역되는 것은 일반적이다. 예를 들어, 컴퓨터 회사 Fujitsu는 여러 해 동안 영어를 사용하는 나라에서 Facom이라는 이름을 사용하였다. 다수의 문자 세트를 가진 언어에서 문제가 악화될 수 있다. 일본 회사는 가다까나로 된 하나의 이름 버젼과, 히라가나로 된 하나의 이름 버젼을 가질 수도 있다.

이러한 이유들과 보다 많은 이유 때문에, 이름 및 설명 객체는 단일 객체에 대해 여러 번 나타날 수 있다. 각각의 인스턴스는 언어 코드가 붙여진다. UDDI 버젼 3에서는 동일한 언어 코드를 가진 다수의 인스턴스가 있을 수 있다(이것은 버젼 2에서는 허용되지 않는다).

발견 한정사는 혼란을 추가한다. 전술한 바와 같이, UDDI 검색은 사례에 민감한 검색과 사례에 민감하지 않은 검색 모두를 지원해야 하며, 이것은 X.500 디렉토리에 데이타를 두번 저장함으로써 최상으로 처리된다.

아래의 예는 추상 클래스, 및 기업 엔티티의 이름들의 집합에 대해 사용되는 도출된 클래스들 중 하나인 euBusinessEntityName를 나타낸다.

```
set object-class uddiObjectClass:202 =
{       # abstract class as parent for all names
        name = euName
        subclass-of top
        must-contain
                euNameKey
        may-contain
                euLanguage
};

set object-class uddiObjectClass:331 =
{       # name of a Business Entity
        name = euBusinessEntityName
        subclass-of euName
        must-contain
                euBusinessEntityNameValue,
                euBusinessEntityNameValueIC
        # inherits euNameKey and euLanguage from euName
};
```

euBusinessEntityNameValue는 이름의 사례 민감 버젼을 포함하는 속성인 반면, euBusinessEntityNameValueC는 "사례 무시"로서 표시되는 버젼이며, 따라서 사례에 민감하지 않다는 점에 유의한다. 추상 클래스로부터 상속되는 euNameKey 필드는 이름들의 순서화를 제어하는 데 사용되며, 고유한 명명 속성을 제공한다.

이름 객체의 일례는 다음과 같을 수 있다:

```
euNameKey = 890abcde-890a -890a -890a -890abcdef123
euLanguage = EN
euBusinessEntityNameValue = McKenna's Validation Systems
euBusinessEntityNameValueIC = McKenna's Validation Systems
```

다시, 도 15는 기업 엔티티에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 나타내고 있지만, 이것은 추상 클래스에서 비교적 복잡한 객체의 표현을 위해 본 발명의 일 실시예에 따라 계층 구조를 하위 구조에 도입하는 일례를 동일하게 나타내고 있다. 도 15의 기업 엔티티(151)는 추상 클래스에 동일하게 적용될 수 있으며, 결합 템플릿의 다가 요소들은 추상 클래스(151)의 자식들(152, 153)로서 표현된다. 자식들이 없을 수도 더 있을 수도 있다.

또 하나의 문제는 사용자가 그의 제어 하에 기업 엔티티들만을 변경할 수 있어야 하는 요건의 효율적인 구현을 생성하는 것에 관한 것이다. 본 발명의 일 실시예에 따르면, 이것은 사용자 객체의 사용자의 자식들에 의해 기업 엔티티들이 제어되도록 함으로써 해결될 수 있다. 이것은 보다 쉽게 보안을 구현할 수 있게 해준다.

공표 사용자만이 그가 소유한 정보를 변경할 수 있도록 보장하는 것이 중요할 수 있다. 이것은 다양한 설계에 의해 이루어질 수 있다. 그러나, 최적의 설계는 사용자가 하나의 항목을 공표할 자격이 있는지, 즉 소정의 사용자에 의해 제어되는 모든 데이타가 그 사용자의 서브 트리 내에 위치하는지를 즉시 명백하게 만든다.

이러한 설계 결정은 전체적으로 기업 엔티티들에 대한 액세스의 용이성에 영향을 주지 않는데, 이는 기업 엔티티들로의 모든 조회가 일반성 또는 성능의 손실 없이 계층 구조 내의 사용자 레벨 위로부터 행해질 수 있기 때문이다.

또 하나의 문제는 findRelatedBusiness 메서드의 구현과 특히 관련하여 공표자 표명의 효율적인 구현을 생성하는 것에 관한 것이다. 본 발명의 일 실시예에 따르면, 이것은 기업 객체의 기업 자식들에 관한 공표자 표명을 만듦으로써 해결될 수 있다. 이것은 그러한 기준을 검색할 필요성을 없앤다.

하나의 주요한 공표자 표명의 사용은 find_RelatedBusinesses 조회에 있다. 이 조회는 특정 기업 엔티티를 지정하고, 완성된 공표자 표명에 의해 그 엔티티와 관련된 모든 기업 엔티티들에 대한 정보를 요구한다. 이 조회는 공표자 표명을 그와 관련된 기업 엔티티의 아래에 배치하는 계층 구조에 의해 간단해지고 가속된다. 이것은 일관성을 증가시키는 추가 이익을 갖는다. 기업 엔티티가 삭제될 때, 모든 관련 공표자 표명(이제는 무관해진)이 그와 함께 삭제된다.

또 하나의 문제는 사용자가 그의 제어 하에 T모델들만을 변경할 수 있어야 하는 요건의 효율적인 구현을 생성하는 것과 관련된다. 본 발명의 일 실시예에 따르면, 시스템은 사용자에 의해 정의되는 T모델들을 사용자 객체의 자식들로 간주한다. 이것은 쉽게 보안을 구현할 수 있게 해준다.

사용자 엔트리 아래에 기업 엔티티를 배치하는 것을 지배하는 것들과 유사한 이유로, 사용자 정의 T모델들을 이들을 정의하는 사용자의 사용자 엔트리 아래에 배치하는 것은 현명한 것이다. T모델들을 찾는 것에 나쁜 영향이 없는데, 이는 T모델들이 모두 고유하게 명명되므로 단일 인덱스 액세스를 통해 찾아질 수 있기 때문이다.

또 하나의 문제는 관계에 의한 공표자 표명의 효율적인 검색을 구현하는 것에 관한 것이다. 본 발명의 일 실시예에 따르면, 이것은 관계 키를 갖춘 참조를 공표자 표명 엔트리의 보조 클래스로 간주함으로써 해결될 수 있다. 키를 갖춘 참조가 자식(하나의 구현)인 경우, 이것은 동일한 효율로 검색될 수 없으며, 관계에 대한 검색은 상태 상의 (중요한) 필터와 같은 공표자 표명의 내용 상의 검색과 결합될 수 없다(완성된 표명들만이 고려된다).

X.500 스키마 시스템은 다른 객체 클래스들을 데이타 요소로서 포함하는 객체 클래스들의 구축을 지원할 수 없다. 예를 들어, 키를 갖춘 참조는 공표자 표명의 데이타 요소일 수 없다. 키를 갖춘 참조를 공표자 표명의 자식으로 간주하는 것이 가능하지만, 이것은 키를 갖춘 참조의 내용들을 참조하는 효율적인 검색의 구축을 용이하게 하지 못한다.

키를 갖춘 참조를 공표자 표명 엔트리에 대한 보조 클래스로 간주하는 것은 문제에 대한 효율적인 솔루션이다. 게다가, 키를 갖춘 참조가 표명의 일부인 것처럼 그 내용을 검색하는 것이 가능하다.

전술한 바와 같이, 공표자 표명의 일례는 다음과 같을 수 있다:

euPublisherAssertionKey = 7890abcd-7890-7890-7890-7890abcdef12

euFromBusinessKey = 34567890-3456-3456-3456-34567890abcd

euFromUserKey = 23456789-2345-2345-2345-234567890abc

euToBusinessKey = 09876543-6543-6543-6543-dcba09876543

euToUserKey = 98765432-5432-5432-5432-cba098765432

euPublisherAssertionStatus = status:complete

euPublisherAssertionTModel=uuid:807A2C6A-EE22-470D-ADC7-E0424A337C03

euPublisherAssertionKeyName = wholly-owned subsidiary

euPublisherAssertionKeyValue = parent-child

보조 객체 클래스는 euPublisherAssertionKeyReference이며, 위에 열거된 최종 3개의 속성은 이 클래스의 데이타 요소들이다.

본 발명의 일 실시예에 따르면, 컴퓨터 어쏘시에이츠에 의한 eTrust™ 디렉토리와 같은 디렉토리는 이상적인 기업 UDDI 레지스트리 플랫폼을 구현하는 데 사용될 수 있다. 완전 준수 LDAPv3, X.500 전자 디렉토리인 eTrust 디렉토리는 UDDI 웹 서비스 구현을 지원하기 위해 사용될 수 있다. eTrust 디렉토리는 UDDI 구현이 기업에 중요한 대규모의 디렉토리 서비스 애플리케이션에서 양호하게 입증된 매우 완성도가 높은 디렉토리 솔루션을 도입하는 것을 허용한다.

eTrust 디렉토리가 UDDI 레지스트리를 구축할 플랫폼으로 매우 매력적이게 하는 eTrust 디렉토리의 많은 고유 기능들이 있다. 이들 중 일부는 액세스 제어 정책, 역할, 보안 프록시, 상호 인증, 분산 인증, 분산 SSL 인증서 주체 검증 및 네트워크 어드레스 검증을 포함하는 보안 기능; 병렬 분산 검색, 부하 공유, 질의 스트리밍 및 최단 경로 라우팅을 포함하는 분산 및 라우팅 능력; 재생 기반 메카니즘(복수 기입으로 알려짐)의 속도 및 효율을 상태 기반 복원 및 조정 기술과 결합시키는 멀티 마스터 복제 스킴; 데이타 베이스들의 핫 스왑, 네트워크 페일오버 및 디렉토리 시스템 에이전트(DSA) 페일오버를 포함하는 이용성 기능; 고속으로 간주되는 캐싱 설계; 및 동적 구성(데이타 타입, 스키마 규칙, 보안, 지식 등), 무제한의 데이타 크기, 일반 정보 완전성 규칙, 광범위한 관리 제어 및 상호작용 명령 컨솔을 포함하는 전개 기능을 포함한다.

eTrust 디렉토리는 입증된 X.500 디렉토리 솔루션을 제공한다. 이 입증된 토대의 최상부에 완전 표준 준수 UDDI 레지스트리를 인에이블시키는 UDDI 시맨틱 브리지가 구현될 수 있다. 기반 디렉토리 솔루션의 능력으로 인해, 본 명세서에 개시된 실시예들은 기존 UDDI 표준에 대한 변경 또는 확장을 요구하지 않고 유연한 보안성, 분산 및 관리성을 전달할 수 있다.

본 실시예의 하나의 문제는 디렉토리의 다른 섹션들에 저장된 엔티티들 사이의 관계를 어떻게 맵핑할 것인지를 처리하는 것이다.

UDDI 데이타 구조는 주로 계층적이지만, 상이한 객체들 사이의 교차 관계에 문제가 있을 수 있다.

본질적으로 두 카테고리의 관계, 즉 대체 이름 및 교차 관계가 있다. 본 발명의 일 실시예에 따르면, 이 문제는 별명의 개념을 이용하여 대체 이름을 해결함으로써 해결된다. 본질적으로, 이것은 외부 엔티티를 주요 엔티티의 가상 자식으로서 첨부하는 효과를 갖는다.

본 실시예는 고유 키들을 이용하여 교차 관계의 문제를 해결한다. 본질적으로, 이것은 계층 구조 디렉토리 시스템 내의 연결되지 않은 서브 트리들 사이에 존재하는 데이타 엔티티들 사이의 관계를 모델링하는 RDBMS 기술에서의 주요/외부 키 시스템과 같은 '관계 포인터'를 생성하는 효과를 갖는다.

본 발명의 실시예들에 따른 별명의 사용이 이제 설명된다. 제1 시나리오는 UDDI 기업 서비스 투영의 구현에 의해 가장 명확하게 증명된다. 기업 서비스 투영은 사실상 기업 서비스의 대체 이름이다. 기업 서비스 투영은 기업 A에 속하는 것으로 보이지만 사실은 기업 B에 의해 소유되고 정의되는 기업 서비스이다.

도 5를 참조하면, 기업 A에 의해 소유되는 서비스인 기업 서비스(51)는 기업 B에 속하는 것으로도 보인다. 기업 A에 의해 기업 서비스(51)에 만들어진 임의의 변화는 기업 B 아래에 나타나는 투영된 서비스에 반영된다. 마찬가지로, 기업 서비스(51)가 레지스트리로부터 삭제되는 경우, 그것은 더 이상 기업 A 또는 기업 B의 아래에 나타나지 않는다. 또한, 기업 엔티티 B는 기업 서비스(51)를 편집하거나 변경할 수 없다. 편집 및 모든 다른 공표 목적을 위해, 기업 A만이 기업 서비스(51)에 액세스한다.

디렉토리 별명 시스템은 이러한 효과를 달성하기 위해 사용될 수 있다. 기업 서비스(51)의 별명이 기업 엔티티 B에 추가된다. 별명은 실제로 '누군가 이 별명을 볼 때 그들에게 여기 위의 이 다른 엔트리를 보여준다'고 말하는 디렉토리 서버에 대한 특수 표시자이다.

이것은 오리지날 서비스가 편집될 때 그 변경도 투영에서 보일 수 있다는 것을 의미한다. eTrust 디렉토리의 경우와 같이, 디렉토리 시스템이 별명 완전성을 지원하는 경우, 서비스가 삭제되면, 그 투영도 자동으로 삭제된다.

또한, 디렉토리 서버는 검색되는 투영된 기업 서비스를 각각의 부모 아래에서 한 번씩 두번 볼 수 있도록 구성될 수 있다. 이것은 기업 서비스의 부모들을 분석하는 것이 필요한 검색을 행할 때 유용할 수 있다.

몇몇 상황은 디렉토리 계층 구조의 연결되지 않은 부분들 내의 객체들이 관계를 유지하는 것을 요구한다.

이것의 일례는 결합 템플릿들과 T모델들 사이이다. T모델들은 다양한 목적을 위해 UDDI 전반에서 사용된다. 이들은 카테고리화 키, 검색 식별자, (UDDI) 관계 설명자, 및 이 예에서 기술 사양 '지문'이다. 결합 템플릿에 첨부되는 T모델은 결합 템플릿(도 8 참조)이 따르는 기술 사양을 설명한다. 예를 들어, 공표자는 그들의 결합 템플릿이 SOAP 1.1 표준을 따른다는 것을 표명하는 T모델을 첨부할 수 있다.

레지스트리는 일반적으로 한정된 일련의 T모델들을 포함하는데, 이들 중 대부분은 수백 개, 심지어 수천 개의 결합 템플릿 엔트리들에 의해 참조된다. 몇몇 경우, 레지스트리는 결합 템플릿의 상세와 함께 임의의 첨부된 T모델의 상세를 반환한다.

본 발명의 일 실시예에 따르면, 관계형 데이타베이스 시스템에서 사용되는 것과 같은 주요/외부 키 시스템이 적적히 수정되어 이용될 수 있다. 레지스트리에 저장된 모든 T모델은 그 자신의 고유(주요) 키를 갖는다. 결합 템플릿은 필요한 T모델의 고유 키와 일치하는 로컬(외부) 키를 추가함으로써 T모델을 참조한다. 도 7은 이것의 일례를 나타낸다. 서버는 T모델 데이타가 결합 템플릿과 함께 반환되는 것이 필요한 경우 당해 T모델을 탐색할 수 있다.

도 6은 결합 템플릿과 T모델 사이의 관계를 나타낸다.

도 7은 2개의 엔티티들 사이의 관계를 T모델 키가 어떻게 생성하는지를 나타낸다.

공표자 표명은 UDDI 저장소의 중요 요소이다. 전술한 바와 같이, 이것은 사용자에게 어느 기업 엔티티들이 관심 있는 기업 엔티티와 관련되어 있으며 이들이 어떻게 관련되어 있는지를 발견하는 능력을 제공한다.

공표자 표명은 남용을 방지할 수 있도록 설계되는데, 표명된 관계는 관련된 양 기업 엔티티들의 소유자가 관계를 표명했었을 때만 보이게 된다. 이러한 보호는 구현을 복잡하게 하고 열악한 성능을 피할 수 있도록 세심한 설계를 필요로 한다는 점에서 비용을 증가시킨다.

하나의 문제는 완전성이다. 공표자 표명은 어느 다른 UDDI 구성보다 복잡한 수명을 갖는다. 이것은 기업 엔티티의 소유자가 그 기업 및 다른 기업 엔티티에 대한 그의 관계에 대한 표명을 만들 때 발생한다. 다른 기업 엔티티의 소유자는 상태 보고를 요구하고, 어떤 표명들이 그들의 기업에 대해 만들어졌는지, 또는 그들이 대역 밖으로 통지될 수 있는지를 알 수 있다. 어느 쪽이나, 다른 기업 엔티티의 소유자는 2개의 기업 엔티티들 사이의 관계에 대해 일치하는 표명을 만들도록 선택할 수 있다. 그 순간에 표명은 완성되며, findRelatedBusinesses를 호출하는 사용자에게 보일 수 있다. 하나 또는 양 표명은 수정 또는 삭제될 수 있으며, 표명은 다시 불완전해져 더 이상 보이지 않아야 한다. 또한, 어느 한 기업 엔티티의 삭제는 표명을 즉시 삭제해야 한다.

공표자 표명 객체는 표명의 완전성을 유지하는 방식으로 관리될 수 있다.

기업 엔티티의 소유자가 이 소유자에 의해 제어되는 기업 엔티티들에 대한 표명을 만들(제거할) 수 있어야 하는 것이 바람직하다.

본 발명의 본 실시예는 X.500 디렉토리에 대해 의도되기는 하지만 UDDI 저장소가 "대부분 판독(read-mostly)" 스토어라는 가정에 기초한다. 이를 위해, 설계는 보다 나은 판독 성능을 위해, 심지어 기입에 보다 무거운 짐을 부과하더라도 최적화된다.

공표자 표명이라고 하는 객체 클래스는 검색 성능을 최적화하기 위한 소망 때문에 UDDI 표준에 의해 요구되는 것 이상의 데이타를 유지하도록 설계된다. 설계는 공표자 표명 상태를 정의하는 조작 속성을 도입한다. 표명의 상태는 디렉토리에 기입할 때 결정되며, 이러한 방법에서는 검색이 행해질 때마다 결정될 필요가 없게 된다.

본 실시예는 또한 사용자 키 형태의 포인터를 사용한다. 공표자 표명이 디렉토리에 기입될 때, 양쪽 기업들에 대한 사용자 키들이 결정되고 객체에 기입된다. 이것은 getAssertionStatusReport 질의를 단순화하는데, 이는 보고를 생성하고 있는 사람의 사용자 키를 포함하는 공표자 표명을 검색하는 것이 그러한 보고를 생성하는 데 필요한 모든 것이기 때문이다.

반대로, 사용자 아래의 모든 기업 키들에 질의한 후 이 기업 키들을 포함하는 공표자 표명들을 찾는 것이 필요한 경우에는 보고를 생성하기 위해 많은 노력이 요구된다.

공표자 표명의 일반적인 하나의 이용은 주어진 기업에 관련된 기업들을 발견하는 것이다. 그러한 질의에 대해 양호한 성능을 제공하기 위하여, 기업에 관련된 공표자 표명은 기업의 자식 노드로서 배치된다.

또한, 각 표명의 상태는 조작 속성으로서 표명에 기록된다. 이것은 관심 있는 회사의 아래에 위치하는 완전한 상태를 가진 공표자 표명에만 질의하는 것을 가능하게 한다. 이것은 findRelatedBusinesses에 대한 검색을 단순화하는데, 이는 이러한 검색이 완전한 표명만을 재호출하기 때문이다.

보안을 간략화하기 위하여, 사용자에 의해 제어되는 모든 기업들 및 그들의 공표자 표명은 그 사용자의 계정 엔트리 아래의 자식 노드들일 수 있다. 이러한 구현은 단지 사용자의 계정 엔트리 아래의 서브 트리에 대한 사용자 액세스를 허용함으로써 액세스 제어를 실현한다.

상태를 표현하는 조작 속성은 UDDI 구현에 의해 관리된다는 점에 유의한다. 사용자가 표명된 다른 기업에 의해 이미 표명된 표명을 공표할 때, UDDI 구현은 다른 기업의 사용자에 의해 제어되는 또 하나의 서브 트리 내에 있는 다른 표명의 상태를 갱신한다. 액세스 제어는 이것을 가능하게 한다.

관련된 2개의 기업 엔티티들 각각 아래에 하나씩인 2개의 공표자 표명 객체를 저장하는 다른 실시예로서, 단일 공표자 표명 객체가 그 자신의 서브 트리 내에 제공된다. 예를 들어, 공표자 표명 서브 트리는 저장소 객체 아래 제공될 수 있다. 이 경우에 표명이 최초로 저장될 때, 표명은 불완전한 상태가 주어진다(예를 들어, 어느 쪽이 표명했는가에 따라 tokeyincomplete 또는 fromkeyincomplete). 공표자 표명이 상보적인 사용자에 의해 표명되는 경우, 상태는 완전한 것으로 변경된다. 공표자 표명이 둘 중의 하나에 의해 삭제되는 경우, 상태는 불완전한 것으로 다시 변경된다. 공표자 표명이 양쪽에 의해 삭제되는 경우, 공표자 표명 객체는 삭제된다. 이롭게도, 이것은 단지 하나의 표명 사본이 있게 하며, 유지 작업의 대부분은 표명의 상태를 유지하는 단일 속성의 수정을 행하는 것으로 이루어진다.

도 12는 본 발명의 일 실시예에 따른 계층 구조를 개략적으로 도시하고 있다. 이 개략도는 공표자 표명 객체가 기업 엔티티 및/또는 저장소 객체 아래에 배치되는 양 대안들을 도시하고 있다.

도 8은 공표자 표명을 추가하도록 요구하는 방법을 나타낸다. 단계 S80에서, 요구가 유효한지에 대한 결정이 이루어진다. 유효하지 않은 경우(No, 단계 S80), 요구는 실패한다(단계 S92). 요구가 유효한 경우(Yes, 단계 S80), 요구가 기점 기업 소유자의 것(from business ours)인지에 대한 결정이 이루어진다(단계 S82). 요구가 기점 기업 소유자의 것이 아닌 경우(No. 단계 S82), 요구가 종점 기업 소유자의 것(to business ours)인지에 대한 결정이 이루어진다(단계 S84). 종점 기업의 것이 아닌 경우(No, 단계 S84), 요구는 실패한다(단계 S92). 요구가 종점 기업의 것인 경우(Yes, 단계 S84), 표명이 기점 소유자에 의해 이루어진 것인지에 대한 결정이 이루어진다(단계 S86). 표명이 기점 소유자에 의해 이루어진 것이 아닌 경우(No, 단계 S86), 불완전한 표명이 기입된다(단계 S94). 표명이 기점 소유자에 의해 만들어진 경우(Yes, 단계 S86), 완전한 표명이 기입된다(단계 S96). 단계 S82로 돌아가서, 요구가 기점 기업 소유자의 것으로 결정되면(Yes, 단계 S82), 요구가 종점 기업 소유자의 것인지에 대한 결정이 이루어진다(단계 S88). 종점 기업 소유자의 것이 아닌 경우(No, 단계 S88), 표명이 종점 소유자에 의해 이루어진 것인지에 대한 결정이 이루어진다(단계 S90). 표명이 종점 소유자에 의해 이루어지지 않은 경우(No, 단계 S90), 불완전한 표명이 기입된다(단계 S94). 단계 S88의 결과가 Yes(종점 기업 소유자)인 경우, 또는 단계 S90의 결과가 Yes(종점 소유자에 의해 이루어진 표명)인 경우, 완전한 표명이 기입된다(단계 S96).

그 다음 문제는 검색 동작 동안 중간 검색 결과 모음의 구축을 어떻게 최적화함으로서 디렉토리 저장 매체 제한을 고려하여 디렉토리 액세스 및 반복 인-메모리 동작 양자를 최소화하는 것이다. 실제로, 디렉토리 엔트리들은 임의의 순서로 저장되고 반환될 수 있으며, 디렉토리 결과들은 너무 커서 분류하지 못할 수도 있다.

본 발명의 일 실시예에 따르면, 현저한 이름에 의해 중간 결과들을 분류하는 고유 결과 분류 스킴과 결합된 객체 지향 인-메모리 데이타 저장 시스템이 제공된다. 이것은 하나의 검색이 많은 상이한 타입의 객체들-기업 엔티티, 기업 서비스 등-을 반환하는 것을 가능하게 하며, 또한 시스템이 데이타를 사용자에게 반환하기 위한 올바른 XML 구조를 쉽게 구축할 수 있게 해준다. 웹 서비스 상호작용은 XML로 되어 있다는 점에 유의한다.

이러한 시스템이 이제 설명된다. 본 발명의 UDDI 기업 엔티티 및 임의의 자식 데이타 요소들은 아래의 계층 구조에 따르는 디렉토리 내에 (부분적으로) 표현된다:

BusinessEntity

- BusinessService
  - BindingTemplate
  - BindingTemplate
  - ServiceName
  - ServiceName
- BusinessService
  - BindingTemplate
  - BindingTemplate
  - ServiceName
  - ServiceName
- BusinessName
- BusinessName
- BusinessDescription
- BusinessDescription

서비스 이름, 기업 이름 및 기업 설명은 하위 구조 및 객체 분할을 다루는 본 발명의 양태들과 관련하여 설명되었다는 점에 유의한다.

기업 엔티티 검색 코드는 필요한 기업 엔티티 또는 기업 엔티티들의 고유 키에 기초하여 디렉토리 서브 트리 검색을 수행한다. 이 검색은 발견된 엔트리들과 모든 서브 엔트리들을 반환한다. 디렉토리 표준은 반환된 엔트리들에 대한 임의의 특정 순서를 보증하지 않거나, 심지어 서브 엔트리들은 그들의 부모 엔트리를 바로 뒤따른다.

따라서, 검색 코드는 반환된 엔트리들을 현저한 이름에 의해 분류한다. 이것은 서브 엔트리들이 그들의 부모 뒤에 순서화되고, 부모-자식 관계가 쉽게 구별될 수 있음을 보증한다. 다양한 분류 알고리즘이 이용될 수 있다. 이용되는 분류 알고리즘은 엔트리들이 부분적으로 분류되는 경우에 고성능의 특성을 나타내야 한다.

결과 구축을 위한 알고리즘은 본질적으로 동작에 있어서 '깊이 우선의 좌에서 우로의 트리 걷기(depth-first, left-to-right tree-walk)'이다. 이것은 다르게는 그래프 이론에서 '후순서 횡단(postorder traversal)'으로 알려져 있다.

분류된 리스트는 새로운 기업 엔티티 객체의 구축자 메서드로 전달된다. 이 객체는 예를 들어 UDDI 기업 엔티티를 표현하도록 설계된 객체 지향 프로그래밍 구조일 수 있다. 기업 엔티티 객체는 엔트리 리스트에 제공된 데이타로부터 그 자신을 구축하기 위한 코드를 포함한다. 이 코드는 리스트를 통해 반복 이동하여 각각의 엔트리에 대한 결정을 행한다. 리스트 내의 제1 엔트리는 기업 엔티티 자체에 대한 주 엔트리이어야 하는 것으로 이해되며, 코드가 다른 기업 엔티티를 발견하자마자 구축이 완료된 것으로 이해되는데, 리스트의 순서화가 이것을 보장한다. 코드가 기업 서비스 또는 다른 자식 엔트리를 발견하자마자, 적당한 타입의 객체가 인스턴스화되며, 리스트는 리스트의 어디에서 시작해야 할지를 알리는 포인터와 함께 새로운 객체의 구축자에게 전달된다.

각 객체는 그 자체의 구축을 처리하고 임의의 자식 엔트리들의 구축을 적당한 자식 객체들에 위임하는 본질적으로 유사한 프로세싱 코드를 포함한다.

이러한 방식으로, 단일 디렉토리 검색만이 수행될 필요가 있으며, 결과 리스트는 효율적인 방식으로 처리되고, 모든 엔트리는 한번 처리된다. 리스트가 임의의 순서로 남겨진 경우, 또는 소정의 다른 방식으로 저장된 경우, 리스트는 결과 엔트리들로부터 UDDI 계층 구조를 올바르게 구축하기 위해 다수의 전달에서 처리되어야 한다.

구축 및 리스트 프로세싱의 계층 구조 내 상이한 프로그래밍 객체들로의 위임은 프로세싱 코드를 비교적 작은 크기로 유지하여 보다 효율적으로, 궁극적으로 보다 빠르게 만든다.

도 9는 분류된 엔트리 리스트의 표현을 포함하는 프로그래밍 구조(객체)를 나타낸다. 항목들의 리스트 내에 임의의 추가 항목들이 존재하는지에 대한 결정이 이루어진다. 추가적인 항목이 없는 경우(No, 단계 S100), 프로세스는 종료한다(단계 S118). 추가 항목이 존재하는 경우(Yes, 단계 S100), 리스트 내의 다음 항목이 검색된다(단계 S102). 이어서, 항목이 이 객체 타입인지에 대한 결정이 이루어진다. 항목이 이 객체 타입인 경우(Yes, 단계 S104), 항목에 기초하여 객체 속성이 설정되며(단계 S106), 프로세스는 단계 S100으로 돌아간다. 항목이 이 객체 타입이 아닌 경우(No, 단계 S104), 이 객체 타입의 항목이 이미 처리되었는지에 대한 결정이 이루어진다(단계 S108). 이 객체 타입의 항목이 아직 처리되지 않은 경우(No, 단계 S108), 프로세스는 단계 S100으로 돌아간다. 이 객체 타입의 항목이 처리된 경우(Yes, 단계 S108), 항목이 이 객체의 고유 컴포넌트(예를 들어, 이름, 설명 등)인지에 대한 결정이 이루어진다. 항목이 고유 컴포넌트인 경우(Yes, 단계 S110), 항목은 객체 속성에 추가되며, 추가 프로세싱이 행해질 수 있고(단계 S112), 프로세스는 단계 S100으로 돌아간다. 항목이 고유 컴포넌트가 아닌 경우(No, 단계 S110), 항목이 이 객체의 자식 객체(예를 들어, 이 객체가 기업 엔티티인 경우 기업 서비스)인지에 대한 결정이 이루어진다. 항목이 자식 객체인 경우(Yes, 단계 S114), 시스템은 정확한 타입의 객체를 인스턴스화하고, 항목들의 리스트를 구축자에게 전달하며(단계 S116), 프로세스는 단계 S100으로 돌아간다. 항목이 자식 객체가 아닌 경우(No, 단계 S114), 프로세스는 단계 S100으로 돌아간다.

아래의 '실제 워드' 예는 LDAP 디렉토리가 반환할 것으로 예측될 수 있는 임의의 순서화의 종류를 나타낸다.

```
SearchResultEntry
 objectName:
businessKey=1ba3034aeef738da00eef78599fe0004,userKey=1ba3034aedb915
4900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessEntity
  type: businessKey
  value: 1ba3034aeef738da00eef78599fe0004

SearchResultEntry
 objectName:
descriptionKey=1ba3034aeef738da00eef786302b0008,businessKey=1ba3034
aeef738da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o
=CA
 attributes
  type: objectClass
  value: uddiDescription

SearchResultEntry
 objectName:
serviceKey=1ba3034aeef738da00eef789707f000c,businessKey=1ba3034aeef
738da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessService

SearchResultEntry
 objectName:
nameKey=1ba3034aeef738da00eef78970da000d,serviceKey=1ba3034aeef738d
a00eef789707f000c,businessKey=1ba3034aeef738da00eef78599fe0004,userKe
y=1ba3034ae
db9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessServiceName

SearchResultEntry
 objectName:
bindingKey=1ba3034aeef738da00eef7899fb7000e,serviceKey=1ba3034aeef7
38da00eef789707f000c,businessKey=1ba3034aeef738da00eef78599fe0004,user
Key=1ba303
4aedb9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: bindingTemplate

SearchResultEntry
 objectName:
nameKey=1ba3034aeef738da00eef7862fe50007,businessKey=1ba3034aeef7
38
da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o=CA
attributes
 type: objectClass
 value: businessEntityName
```

리스트 1 - 굵은 글자로 된 이름 엔트리는 리스트의 최상부에 있는 기업 엔티티 엔트리의 일부이며, 기업 서비스 엔트리 및 기업 엔티티의 다른 분기 자식들의 앞에 나타난 경우 유용하다. 그러나, 이것은 리스트의 끝에 나타나, 기업 엔티티의 모든 직계 자식들이 처리되었음을 보장할 수 있도록 임의의 프로세싱 코드가 전체 리스트를 검색하도록 강제한다. 이것은 가장 효율적이 아닐 수 있다.

따라서, 본 발명의 일 실시예에 따라 형성된 규칙들에 따라 분류된 동일 데이타의 한 버전은 다음과 같다:

```
SearchResultEntry
 objectName:
businessKey=1ba3034aeef738da00eef78599fe0004,userKey=1ba3034aedb915
4900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessEntity
  type: businessKey
  value: 1ba3034aeef738da00eef78599fe0004

SearchResultEntry
 objectName:
descriptionKey=1ba3034aeef738da00eef786302b0008,businessKey=1ba3034
aeef738da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o
=CA
 attributes
  type: objectClass
  value: uddiDescription

SearchResultEntry
 objectName:
nameKey=1ba3034aeef738da00eef7862fe50007,businessKey=1ba3034aeef7
38
da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessEntityName

SearchResultEntry
 objectName:
serviceKey=1ba3034aeef738da00eef789707f000c,businessKey=1ba3034aeef
738da00eef78599fe0004,userKey=1ba3034aedb9154900edb915491c0001,o=CA
 attributes
 type: objectClass
 value: businessService

SearchResultEntry
 objectName:
bindingKey=1ba3034aeef738da00eef7899fb7000e,serviceKey=1ba3034aeef7
38da00eef789707f000c,businessKey=1ba3034aeef738da00eef78599fe0004,user
Key=1ba303
4aedb9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: bindingTemplate

SearchResultEntry
 objectName:
nameKey=1ba3034aeef738da00eef78970da000d,serviceKey=1ba3034aeef738d
a00eef789707f000c,businessKey=1ba3034aeef738da00eef78599fe0004,userKe
y=1ba3034ae
db9154900edb915491c0001,o=CA
 attributes
  type: objectClass
  value: businessServiceName
```

리스트 2 - 굵은 글자로 표시된 엔트리는 이제 리스트 내의 보다 논리적인 장소에 나타나며, 프로세싱 코드는 이제 이것을 이용하여 기입될 수 있다. 엔트리의 수가 실제적인 서버 로드로 증가할 때, 프로세싱 시간의 절약은 중요할 수 있다.

다음은 본 발명의 다른 실시예이다.

```
#       schema for representing UDDI data and / or relationships in a Directory
                                                                  .....expression 100

# Computer Associates eTrust UDDI Configuration Schema
# Copyright (c) 2002 Computer Associates Inc


set oid-prefix uddiAttributeType = (1.3.6.1.4.1.3327.80.1);
set oid-prefix uddiObjectClass = (1.3.6.1.4.1.3327.80.2);
set oid-prefix uddiBinding = (1.3.6.1.4.1.3327.80.3);


# ------------------------------------------------------------------------
```

```
# Key attributes

set attribute uddiAttributeType:201 =
{ # used in KeyedReference and all its derived classes
   name = euKeyedReferenceKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:202 =
{ # used in UserAccount
   name = euUserKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:203 =
{ # used in BusinessEntity, TModel, possibly others
   name = euParentUserKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:204 =
{ # used in BusinessEntity
   name = euBusinessEntityKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:205 =
{ # used in BusinessService, possibly others
   name = euParentBusinessKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:206 =
```

```
{ # used in BusinessService
   name = euBusinessServiceKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:207 =
{ # used in BindingTemplate, possibly others
   name = euParentServiceKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:208 =
{ # used in BindingTemplate
   name = euBindingTemplateKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:209 =
{ # used in TModel
   name = euTModelKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:210 =
{ # used in PublisherAssertion
   name = euPublisherAssertionKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:211 =
{ # used in PublisherAssertion
   name = euFromBusinessKey
   syntax = caseIgnoreString
```

```
    single-valued
};
set attribute uddiAttributeType:212 =
{ # used in PublisherAssertion
    name = euFromUserKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:213 =
{ # used in PublisherAssertion
    name = euToBusinessKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:214 =
{ # used in PublisherAssertion
    name = euToUserKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:216 =
{ # used in DiscoveryURL
    name = euDiscoveryURLKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:217 =
{ # used in Contact
    name = euContactKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:218 =
```

```
{ # used in Address
   name = euAddressKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:219 =
{ # used in Address
   name = euAddressTModelKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:220 =
{ # used in AddressLine
   name = euAddressLineKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:221 =
{ # used in Phone
   name = euPhoneKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:222 =
{ # used in Email
   name = euEmailKey
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:223 =
{ # used in TmodelInstanceInfo
   name = euInstanceTModelKey
   syntax = caseIgnoreString
```

```
    single-valued
};
set attribute uddiAttributeType:224 =
{ # used in Name, and all derived classes
    name = euNameKey
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:225 =
{ # used in Decription, and all derived classes
    name = euDescriptionKey
    syntax = caseIgnoreString
    single-valued
};

# ----------------------------------------------------------------------------
# Attributes used in keyed references

set attribute uddiAttributeType:301 =
{ # used in BusinessEntityCategory
    name = euBusinessEntityCategoryKRTModel
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:302 =
{ # used in BusinessEntityCategory
    name = euBusinessEntityCategoryKRKeyName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:303 =
{ # used in BusinessEntityCategory
    name = euBusinessEntityCategoryKRKeyValue
```

```
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:304 =
{ # used in BusinessEntityIdentifier
    name = euBusinessEntityIdentifierKRTModel
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:305 =
{ # used in BusinessEntityIdentifier
    name = euBusinessEntityIdentifierKRKeyName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:306 =
{ # used in BusinessEntityIdentifier
    name = euBusinessEntityIdentifierKRKeyValue
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:307 =
{ # used in BusinessServiceCategory
    name = euBusinessServiceCategoryKRTModel
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:308 =
{ # used in BusinessServiceCategory
    name = euBusinessServiceCategoryKRKeyName
    syntax = caseIgnoreString
    single-valued
};
```

```
set attribute uddiAttributeType:309 =
{ # used in BusinessServiceCategory
   name = euBusinessServiceCategoryKRKeyValue
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:310 =
{ # used in TModelCategory
   name = euTModelCategoryKRTModel
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:311 =
{ # used in TModelCategory
   name = euTModelCategoryKRKeyName
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:312 =
{ # used in TModelCategory
   name = euTModelCategoryKRKeyValue
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:313 =
{ # used in TModelIdentifier
   name = euTModelIdentifierKRTModel
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:314 =
{ # used in TModelIdentifier
   name = euTModelIdentifierKRKeyName
```

```
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:315 =
{ # used in TModelIdentifier
    name = euTModelIdentifierKRKeyValue
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:316 =
{ # used in PublisherAssertion
    name = euPublisherAssertionKRTModel
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:317 =
{ # used in PublisherAssertion
    name = euPublisherAssertionKRKeyName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:318 =
{ # used in PublisherAssertion
    name = euPublisherAssertionKRKeyValue
    syntax = caseIgnoreString
    single-valued
};

# ------------------------------------------------------------------------
# Attributes used in names and descriptions

set attribute uddiAttributeType:361 =
{ # used in business entity name
```

```
    name = euBusinessEntityNameValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:381 =
{ # used in business entity name
    name = euBusinessEntityNameValueIC
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:362 =
{ # used in business service name
    name = euBusinessServiceNameValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:382 =
{ # used in business service name
    name = euBusinessServiceNameValueIC
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:363 =
{ # used in business entity description
    name = euBusinessEntityDescriptionValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:383 =
{ # used in business entity description
    name = euBusinessEntityDescriptionValueIC
    syntax = caseIgnoreString
    single-valued
```

```
};
set attribute uddiAttributeType:364 =
{ # used in business service description
   name = euBusinessServiceDescriptionValue
   syntax = CaseExactString
   single-valued
};
set attribute uddiAttributeType:384 =
{ # used in business service description
   name = euBusinessServiceDescriptionValueIC
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:365 =
{ # used in tmodel description
   name = euTModelDescriptionValue
   syntax = CaseExactString
   single-valued
};
set attribute uddiAttributeType:385 =
{ # used in tmodel description
   name = euTModelDescriptionValueIC
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:366 =
{ # used in tmodel instance info description
   name = euTModelInstanceInfoDescriptionValue
   syntax = CaseExactString
   single-valued
};
set attribute uddiAttributeType:386 =
{ # used in tmodel instance info description
```

```
    name = euTModelInstanceInfoDescriptionValueIC
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:367 =
{ # used in tmodel instance details description
    name = euTModelInstanceDetailsDescriptionValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:387 =
{ # used in tmodel instance details description
    name = euTModelInstanceDetailsDescriptionValueIC
    syntax = caseIgnoreString
    single-valued.
};
set attribute uddiAttributeType:368 =
{ # used in overview doc description
    name = euOverviewDocDescriptionValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:388 =
{ # used in overview doc description
    name = euOverviewDocDescriptionValueIC
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:369 =

{ # used in Binding Template Description
    name = euBindingTemplateDescriptionValue
    syntax = CaseExactString
```

```
    single-valued
};
set attribute uddiAttributeType:389 =
{ # used in Binding Template Description
    name = euBindingTemplateDescriptionValueIC
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:370 =
{ # used in Contact Description
    name = euContactDescriptionValue
    syntax = CaseExactString
    single-valued
};
set attribute uddiAttributeType:390 =
{ # used in Contact Description
    name = euContactDescriptionValueIC
    syntax = caseIgnoreString
    single-valued
};

# ------------------------------------------------------------------------
# Other attributes

set attribute uddiAttributeType:400 =
{ # used in Name and Description
    name = euLanguage
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:401 =
{ # used in Repository
    name = euRepositoryName
```

```
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:402 =
{ # used in UserAccount
    name = euUserName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:403 =
{ # used in UserAccount
    name = euCredentials
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:404 =
{ # used in UserAccount
    name = euAuthorizedName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:405 =
{ # used in UserAccount and TModel
    name = euHidden
    syntax = boolean
    single-valued
};
set attribute uddiAttributeType:406 =
{ # used in business entity and tmodel
    name = euOperator
    syntax = caseIgnoreString
    single-valued
};
```

```
set attribute uddiAttributeType:407 =
{ # used in contact
   name = euContactName
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:408 =
{ # used in discoveryURL, contact, address, phone, email
   name = euUseType
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:409 =
{ # used in phone
   name = euPhoneNumber
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:419 =
{ # used in email
   name = euEmailAddress
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:411 =
{ # used in address
   name = euSortCode
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:412 =
{ # used in binding template
   name = euHostingRedirector
```

```
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:413 =
{ # used in binding template
    name = euAccessPoint
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:414 =
{ # used in binding template
    name = euAccessPointType
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:415 =
{ # used in tmodel
    name = euTModelName
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:416 =
{ # used in tmodel
    name = euOverviewURL
    syntax = caseIgnoreString
    single-valued
};
set attribute uddiAttributeType:417 =
{ # used in address line
    name = euAddressLineValue
    syntax = caseIgnoreString
    single-valued
};
```

```
set attribute uddiAttributeType:418 =
{ # used in tmodel instance info
   name = euInstanceParms
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:420 =
{ # used in PublisherAssertion
   name = euPublisherAssertionStatus
   syntax = caseIgnoreString
   single-valued
};
set attribute uddiAttributeType:421 =
{ # used in discovery URL
   name = euDiscoveryURLValue
   syntax = caseIgnoreString
   single-valued
};

# ------------------------------------------------------------------------
# Abstract classes - do not attempt to store these in the directory!

set object-class uddiObjectClass:201 =
{ # abstract class as parent for all keyed references
    name = euKeyedReference
    subclass-of top
    kind = abstract
    must-contain
      euKeyedReferenceKey
  # NOTE: a keyed reference should also contain a tModel key, a key name, and
 a key
  # value each derived class adds these, so they can all have different names to
  # facilitate searching the standardised names of these attributes are:
```

```
 # euXXXTModel
 # euXXXKeyName
 # euXXXKeyValue
 # where: XXX is the name of the object and the purpose of the keyed reference
};
set object-class uddiObjectClass:202 =
{ # abstract class as parent for all names
   name = euName
   subclass-of top
   kind = abstract
   must-contain
     euNameKey
   may-contain
     euLanguage
  # NOTE: a name should also have a string containing the name proper, that
string will usually
  # have a name of the pattern euXXXNameValue, where XXX is the name of the
parent object
  # this maximises efficiency of searching
  # there's a second copy of the attribute, with IC appended - this is the ignore
case version
};
set object-class uddiObjectClass:203 =
{ # abstract class as parent for all descriptions
    name = euDescription
    subclass-of top
    kind = abstract
    must-contain
      euDescriptionKey
    may-contain
      euLanguage
  # NOTE: a description should also have a string containing the description
 proper.
```

```
 # that string will usually have a name of the pattern euXXXDescriptionValue,
 # where XXX is the name of the parent object
 # this maximises efficiency of searching
 # there's a second copy of the attribute, with IC appended - this is the ignore
case version

};

# ----------------------------------------------------------------------------
# Keyed Reference types

set object-class uddiObjectClass:301 =
{ # business entity category keyed reference - collection makes up the category
bag
   name = euBusinessEntityCategoryKR
   subclass-of euKeyedReference
   must-contain
    euBusinessEntityCategoryKRKeyValue
   may-contain
    euBusinessEntityCategoryKRTModel,
    euBusinessEntityCategoryKRKeyName
};
set object-class uddiObjectClass:302 =
{ # business entity identifier keyed reference - collection makes up the identifier
bag
   name = euBusinessEntityIdentifierKR
   subclass-of euKeyedReference
   must-contain
    euBusinessEntityIdentifierKRKeyValue
   may-contain
    euBusinessEntityIdentifierKRTModel,
    euBusinessEntityIdentifierKRKeyName
};
```

```
set object-class uddiObjectClass:303 =
{ # business service category keyed reference - collection makes up the category
bag
   name = euBusinessServiceCategoryKR
   subclass-of euKeyedReference
   must-contain
    euBusinessServiceCategoryKRKeyValue
   may-contain
    euBusinessServiceCategoryKRTModel,
    euBusinessServiceCategoryKRKeyName
};
set object-class uddiObjectClass:304 =
{ # tmodel category keyed reference - collection makes up the category bag
   name = euTModelCategoryKR
   subclass-of euKeyedReference
   must-contain
    euTModelCategoryKRKeyValue
   may-contain
    euTModelCategoryKRTModel,
    euTModelCategoryKRKeyName
};
set object-class uddiObjectClass:305 =
{ # tmodel identifier keyed reference - collection makes up the identifier bag
   name = euTModelIdentifierKR
   subclass-of euKeyedReference
   must-contain
    euTModelIdentifierKRKeyValue
   may-contain
    euTModelIdentifierKRTModel,
    euTModelIdentifierKRKeyName
};
set object-class uddiObjectClass:306 =
```

```
{ # publisher assertion keyed reference - used as auxiliary class to give
relationship
   name = euPublisherAssertionKR
   subclass-of euKeyedReference
   kind = auxiliary
   must-contain
    euPublisherAssertionKRKeyValue
   may-contain
    euPublisherAssertionKRTModel,
    euPublisherAssertionKRKeyName
};

# ----------------------------------------------------------------------------
# Name and Description types

set object-class uddiObjectClass:331 =
{ # name of a business entity
   name = euBusinessEntityName
   subclass-of euName
   must-contain
    euBusinessEntityNameValue,
    euBusinessEntityNameValueIC
  # inherits euNameKey and euLanguage from euName
};
set object-class uddiObjectClass:332 =
{ # name of a business service
   name = euBusinessServiceName
   subclass-of euName
   must-contain
    euBusinessServiceNameValue,
    euBusinessServiceNameValueIC
  # inherits euNameKey and euLanguage from euName
};
```

```
set object-class uddiObjectClass:341 =
{ # description of a business entity
    name = euBusinessEntityDescription
    subclass-of euDescription
    may-contain
      euBusinessEntityDescriptionValue,
      euBusinessEntityDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};

set object-class uddiObjectClass:342 =
{ # description of a business service
    name = euBusinessServiceDescription
    subclass-of euDescription
    may-contain
      euBusinessServiceDescriptionValue,
      euBusinessServiceDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:343 =
{ # description of a tmodel
    name = euTModelDescription
    subclass-of euDescription
    may-contain
      euTModelDescriptionValue,
      euTModelDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:344 =
{ # description of a tmodel instance info object
    name = euTModelInstanceInfoDescription
    subclass-of euDescription
    may-contain
```

```
    euTModelInstanceInfoDescriptionValue,
    euTModelInstanceInfoDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:345 =
{ # description of a tmodel instance details object
   name = euTModelInstanceDetailsDescription
   subclass-of euDescription
   may-contain
    euTModelInstanceDetailsDescriptionValue,
    euTModelInstanceDetailsDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:346 =
{ # description of a overview doc object
   name = euOverviewDocDescription
   subclass-of euDescription
   may-contain
    euOverviewDocDescriptionValue,
    euOverviewDocDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:347 =
{ # description of a contact
   name = euContactDescription
   subclass-of euDescription
   may-contain
    euContactDescriptionValue,
    euContactDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};
set object-class uddiObjectClass:348 =
{ # description of a Binding Template
```

```
    name = euBindingTemplateDescription
    subclass-of euDescription
    may-contain
      euBindingTemplateDescriptionValue,
      euBindingTemplateDescriptionValueIC
  # inherits euDescriptionKey and euLanguage from euDescription
};

# ----------------------------------------------------------------------------
# Major objects

set object-class uddiObjectClass:400 =
{ # repository - may be used to break users into groups
    name = euRepository
    subclass-of top
    must-contain
      euRepositoryName
};
set object-class uddiObjectClass:401 =
{ # user account - where we stash our knowledge of the user
    name = euUserAccount
    subclass-of top
    must-contain
      euUserKey,
      euUserName,
      euCredentials
    may-contain
      euAuthorizedName,
      euHidden
  # NOTE: all business entities and tmodels published by this user will be found as
children of this object
};
set object-class uddiObjectClass:402 =
```

```
{ # business entity - details of an entity which provides services
   name = euBusinessEntity
   subclass-of top
   must-contain
    euBusinessEntityKey
   may-contain
    euParentUserKey,
    euAuthorizedName,
    euOperator
 # NOTE: many of the attributes of the business entity are held in children of this
object
 # particularly anything that may occur more than once
};
set object-class uddiObjectClass:403 =
{ # business service - details of a service provided by a business entity
   name = euBusinessService
   subclass-of top
   must-contain
    euBusinessServiceKey
   may-contain
    euParentBusinessKey
 # NOTE: all binding templates for this service will be found as children of this
service
};
set object-class uddiObjectClass:404 =
{ # binding template - details of how to access a particular business service
   name = euBindingTemplate
   subclass-of top
   must-contain
    euBindingTemplateKey
   may-contain
    euParentServiceKey,
    euHostingRedirector,
```

```
    euAccessPoint,
    euAccessPointType
  # NOTE: should have exactly one of either a hosting redirector, or an access
point
};
set object-class uddiObjectClass:405 =
{ # tmodel - a reference to an idea. May be a categorisation scheme, may simply
be a reference to a standard
   name = euTModel
   subclass-of top
   must-contain
    euTModelKey,
    euTModelName
   may-contain
    euAuthorizedName,
    euOperator,
    euOverviewURL,
    euParentUserKey,
    euHidden
  # NOTE: Hidden is used when "deleting" TModels.
};
set object-class uddiObjectClass:406 =
{ # publisher assertion - makes a statement about a relationship between two
businesses
   name = euPublisherAssertion
   subclass-of top
   must-contain
    euPublisherAssertionKey,
    euFromBusinessKey,
    euFromUserKey,
    euToBusinessKey,
    euToUserKey,
    euPublisherAssertionStatus
```

```
  # NOTE: the relationship will be stored as an auxiliary class of type
euPublisherAssertionKeyedReference
  # this allows direct searching for the elements of the auxiliary class
};

# ----------------------------------------------------------------------------
# Minor objects - mostly children of the major objects holding repeating data

set object-class uddiObjectClass:501 =
{ # a discoveryURL - to be found under business entities
    name = euDiscoveryURL
    subclass-of top
    must-contain
      euDiscoveryURLKey,
      euDiscoveryURLValue,
      euUseType
};
set object-class uddiObjectClass:502 =
{ # contact - to be found under business entities - quite complex, with many
possible children
    name = euContact
    subclass-of top
    must-contain
      euContactKey,
      euContactName
    may-contain
      euUseType
};
set object-class uddiObjectClass:503 =
{ # address - to be found under contacts
    name = euAddress
    subclass-of top
    must-contain
```

```
    euAddressKey
  may-contain
    euSortCode,
    euAddressTModelKey,
    euUseType
};
set object-class uddiObjectClass:504 =
{ # address line - to be found under address, making up the lines of the address
  name = euAddressLine
  subclass-of top
  must-contain
    euAddressLineKey,
    euAddressLineValue
};
set object-class uddiObjectClass:505 =
{ # phone - to be found under contacts
  name = euPhone
  subclass-of top
  must-contain
    euPhoneKey,
    euPhoneNumber
  may-contain
    euUseType
};
set object-class uddiObjectClass:506 =
{ # email - to be found under contacts
  name = euEmail
  subclass-of top
  must-contain
    euEmailKey,
    euEmailAddress
  may-contain
    euUseType
```

```
};
set object-class uddiObjectClass:507 =
{ # tmodel instance info - to be found under binding template
   name = euTModelInstanceInfo
   subclass-of top
   must-contain
    euInstanceTModelKey

   may-contain
    euInstanceParms,
    euOverviewURL
};

# ------------------------------------------------------------------------
# Name Bindings.

schema set name-binding uddiBinding:101 =
{ # binding to top - highest level child
   name = euRepository-top
   euRepository allowable-parent top
   named-by euRepositoryName
};
schema set name-binding uddiBinding:102 =
{ # binding to top - highest level child
   name = euUserAccount-top
   euUserAccount allowable-parent top
   named-by euUserKey
};
schema set name-binding uddiBinding:103 =
{ # binding to euRepository
   name = euUserAccount-euRepository
   euUserAccount allowable-parent euRepository
   named-by euUserKey
```

```
};
schema set name-binding uddiBinding:104 =
{ # binding TModel to "top" - used for standard TModels (not published by a user)
   name = euTModel-euRepository
   euTModel allowable-parent euRepository
   named-by euTModelKey
};
schema set name-binding uddiBinding:105 =
{ # binding to organization - highest level child
   name = euRepository-organization
   euRepository allowable-parent organization
   named-by euRepositoryName
};
schema set name-binding uddiBinding:106 =
{ # binding publisher assertions to repository to allow for alternative configuration
   name = euPublisherAssertion-euRepository
   euPublisherAssertion allowable-parent euRepository
   named-by euPublisherAssertionKey
};
schema set name-binding uddiBinding:107 =
{ # binding repository layers - allows multi-level repository structure
   name = euRepository-euRepository
   euRepository allowable-parent euRepository
   named-by euRepositoryName
};
schema set name-binding uddiBinding:201 =
{ # binding business entity to user account - second level
   name = euBusinessEntity-euUserAccount
   euBusinessEntity allowable-parent euUserAccount
   named-by euBusinessEntityKey
};
schema set name-binding uddiBinding:202 =
{ # binding tmodel to user account - second level
```

```
    name = euTModel-euUserAccount
    euTModel allowable-parent euUserAccount
    named-by euTModelKey
};
schema set name-binding uddiBinding:301 =
{ # binding service to business - third level
    name = euBusinessService-euBusinessEntity
    euBusinessService allowable-parent euBusinessEntity
    named-by euBusinessServiceKey
};
schema set name-binding uddiBinding:302 =
{ # binding contact under business - third level
    name = euContact-euBusinessEntity
    euContact allowable-parent euBusinessEntity
    named-by euContactKey
};
schema set name-binding uddiBinding:303 =
{ # binding discoveryURL under business
    name = euDiscoveryURL-euBusinessEntity
    euDiscoveryURL allowable-parent euBusinessEntity
    named-by euDiscoveryURLKey
};
schema set name-binding uddiBinding:304 =
{ # name under business
    name = euBusinessEntityName-euBusinessEntity
    euBusinessEntityName allowable-parent euBusinessEntity
    named-by euNameKey
};
schema set name-binding uddiBinding:305 =
{ # description under business
    name = euBusinessEntityDescription-euBusinessEntity
    euBusinessEntityDescription allowable-parent euBusinessEntity
    named-by euDescriptionKey
```

```
};
schema set name-binding uddiBinding:306 =
{ # publisher assertion under business
   name = euPublisherAssertion-euBusinessEntity
   euPublisherAssertion allowable-parent euBusinessEntity
   named-by euPublisherAssertionKey
};
schema set name-binding uddiBinding:307 =
{ # identifer under business entity
   name = euBusinessEntityIdentifierKR-euBusinessEntity
   euBusinessEntityIdentifierKR allowable-parent euBusinessEntity
   named-by euKeyedReferenceKey
};
schema set name-binding uddiBinding:308 =
{ # category under business entity
   name = euBusinessEntityCategoryKR-euBusinessEntity
   euBusinessEntityCategoryKR allowable-parent euBusinessEntity
   named-by euKeyedReferenceKey
};
schema set name-binding uddiBinding:310 =
{ # description under TModel
   name = euTModelDescription-euTModel
   euTModelDescription allowable-parent euTModel
   named-by euDescriptionKey
};
schema set name-binding uddiBinding:311 =
{ # description of overview URL under TModel
   name = euOverviewDocDescription-euTModel
   euOverviewDocDescription allowable-parent euTModel
   named-by euDescriptionKey
};
schema set name-binding uddiBinding:312 =
{ # identifier under tmodel
```

```
    name = euTModelIdentifierKR-euTModel
    euTModelIdentifierKR allowable-parent euTModel
    named-by euKeyedReferenceKey
};
schema set name-binding uddiBinding:313 =
{ # category under TModel
    name = euTModelCategoryKR-euTModel
    euTModelCategoryKR allowable-parent euTModel
    named-by euKeyedReferenceKey
};
schema set name-binding uddiBinding:401 =
{ # address under contact
    name = euAddress-euContact
    euAddress allowable-parent euContact
    named-by euAddressKey
};
schema set name-binding uddiBinding:402 =
{ # phone number under contact
    name = euPhone-euContact
    euPhone allowable-parent euContact

    named-by euPhoneKey
};
schema set name-binding uddiBinding:403 =
{ # email under contact
    name = euEmail-euContact
    euEmail allowable-parent euContact
    named-by euEmailKey
};
schema set name-binding uddiBinding:404 =
{ # description under contact
    name = euContactDescription-euContact
    euContactDescription allowable-parent euContact
```

```
    named-by euDescriptionKey
};
schema set name-binding uddiBinding:409 =
{ # name under service
    name = euBusinessServiceName-euBusinessService
    euBusinessServiceName allowable-parent euBusinessService
    named-by euNameKey
};
schema set name-binding uddiBinding:410 =
{ # description under service
    name = euBusinessServiceDescription-euBusinessService
    euBusinessServiceDescription allowable-parent euBusinessService
    named-by euDescriptionKey
};
schema set name-binding uddiBinding:411 =
{ # category under service
    name = euBusinessServiceCategoryKR-euBusinessService
    euBusinessServiceCategoryKR allowable-parent euBusinessService
    named-by euKeyedReferenceKey
};
schema set name-binding uddiBinding:412 =
{ # binding templates under services
    name = euBindingTemplate-euBusinessService
    euBindingTemplate allowable-parent euBusinessService
    named-by euBindingTemplateKey
};
schema set name-binding uddiBinding:501 =
{ # lines under addresses
    name = euAddressLine-euAddress
    euAddressLine allowable-parent euAddress
    named-by euAddressLineKey
};
schema set name-binding uddiBinding:502 =
```

```
{ # description under binding template
   name = euBindingTemplateDescription-euBindingTemplate
   euBindingTemplateDescription allowable-parent euBindingTemplate
   named-by euDescriptionKey
};
schema set name-binding uddiBinding:510 =
{ # tmodel instance info under binding template
   name = euTModelInstanceInfo-euBindingTemplate
   euTModelInstanceInfo allowable-parent euBindingTemplate
   named-by euInstanceTModelkey
};
schema set name-binding uddiBinding:601 =
{ # description under tmodel instance info
   name = euTModelInstanceInfoDescription-euTModelInstanceInfo
   euTModelInstanceInfoDescription allowable-parent euTModelInstanceInfo
   named-by euDescriptionKey
};
schema set name-binding uddiBinding:602 =
{ # instance details description under tmodel instance info
   name = euTmodelInstanceDetailsDescription-euTModelInstanceInfo
   euTmodelInstanceDetailsDescription allowable-parent euTModelInstanceInfo
   named-by euDescriptionKey
};
schema set name-binding uddiBinding:603 =
{ # overview doc description under tmodel instance info
   name = euOverviewDocDescription-euTModelInstanceInfo
   euOverviewDocDescription allowable-parent euTModelInstanceInfo
   named-by euDescriptionKey
};
```

본 발명은 본 발명의 본질적인 특성의 사상을 벗어나지 않고 다양한 형태로 구현될 수 있으므로, 전술한 실시예들은 달리 특정되지 않는 한 본 발명을 제한하지 않는다는 것을 이해해야 하며, 첨부된 청구범위에 정의된 바와 같은 본 발명의 사상 및 범위 내에서 넓게 해석되어야 한다. 다양한 변형 및 균등 구성들은 본 발명의 사상 및 범위 및 첨부된 청구범위 내에 포함되는 것으로 의도된다.

(57) 청구의 범위

## 청구항 1.

웹 서비스 장치에서 이용하기 위한 방법으로서,

기업 엔티티 객체(들)를 사용자 객체(들) 아래에 배열하는 단계와,

상응하는 T모델 객체(들)를 사용자 객체(들), 저장소 객체 및 접두사 중 적어도 하나의 아래에 배열하는 단계

를 포함하는 방법.

## 청구항 2.

제 1 항에 있어서,

공표자 표명 객체(들)를 기업 엔티티 객체(들) 아래에 배열하는 단계를 더 포함하는 방법.

## 청구항 3.

제 1 항에 있어서,

서비스 투영 객체(들)을 기업 엔티티 객체(들) 아래에 제공하는 단계를 더 포함하는 방법.

## 청구항 4.

제 3 항에 있어서,

상기 서비스 투영 객체(들)는 별명으로 구현되는 방법.

## 청구항 5.

제 4 항에 있어서,

제 1 필드(들)를 공표자 표명 객체(들)의 속성으로서 제공하는 단계를 더 포함하는 방법.

## 청구항 6.

제 5 항에 있어서,

보조 클래스에 의해서 키 참조(keyed reference)를 나타내는 단계를 더 포함하는 방법.

## 청구항 7.

제 6 항에 있어서,

상기 객체에 대한 소유 및 제어의 체인을 밝히는 객체의 구별 이름(Distinguished Name)을 제공하는 단계를 더 포함하는 방법.

## 청구항 8.

웹 서비스 장치에서 이용하기 위한 컴퓨터 실행가능 명령을 포함하는 컴퓨터 기록 매체로서,

기업 엔티티 객체(들)을 사용자 객체(들) 아래에 배열하는 코드와,

상응하는 T모델 객체(들)를 사용자 객체(들), 저장소 객체 및 접두사 중 적어도 하나의 아래에 배열하는 코드를 포함하는 컴퓨터 기록 매체.

## 청구항 9.

제 8 항에 있어서,

공표자 표명 객체(들)를 기업 엔티티 객체(들) 아래에 배열하는 코드를 더 포함하는 컴퓨터 기록 매체.

## 청구항 10.

제 8 항에 있어서,

서비스 투영 객체(들)을 기업 엔티티 객체(들) 아래에 제공하는 코드를 더 포함하는 컴퓨터 기록 매체.

## 청구항 11.

제 10 항에 있어서,

상기 서비스 투영 객체(들)는 별명으로 구현되는 컴퓨터 기록 매체.

## 청구항 12.

제 11 항에 있어서,

제 1 필드(들)를 공표자 표명 객체(들)의 속성으로서 제공하는 코드를 더 포함하는 컴퓨터 기록 매체.

## 청구항 13.

제 12 항에 있어서,

보조 클래스에 의해서 키 참조(keyed reference)를 나타내는 코드를 더 포함하는 컴퓨터 기록 매체.

## 청구항 14.

제 13 항에 있어서,

상기 객체에 대한 소유 및 제어의 체인을 밝히는 객체의 구별 이름(Distinguished Name)을 제공하는 코드를 더 포함하는 컴퓨터 기록 매체.

요약

사용자 객체 아래에 기업 엔티티 객체를 배열하고, 적어도 하나의 사용자 객체 아래에 상응하는 T모델 객체, 저장소 객체 및 접두사를 배열하는 것을 포함하는 웹 서비스 장치에서 이용하기 위한 방법에 관한 것이다.

대표도

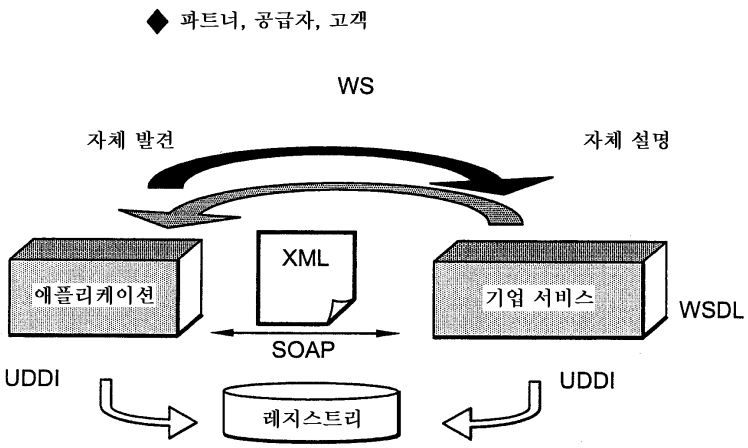
도 4

색인어

웹 서비스 디렉토리, 기업 엔티티 객체, 사용자 객체, 도메인, 템플릿

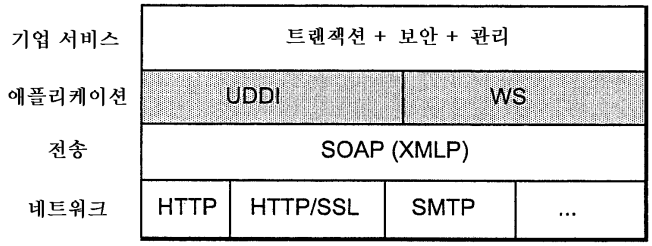
도면

도면1a




웹 서비스 관련 기술

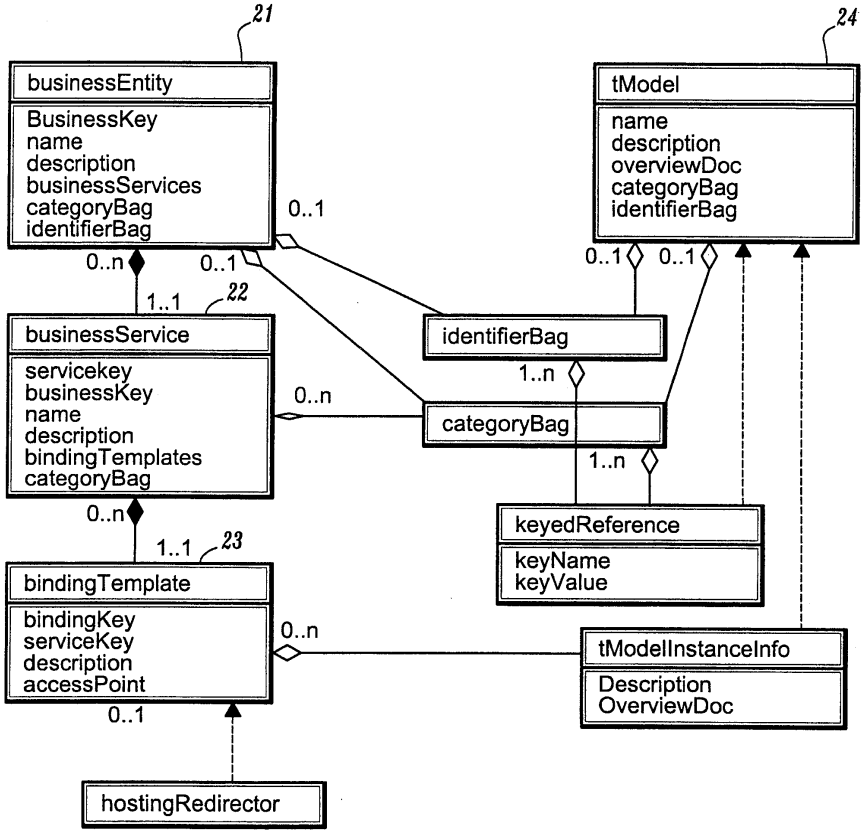
도면1b




WS 프로토콜 스택 관련 기술

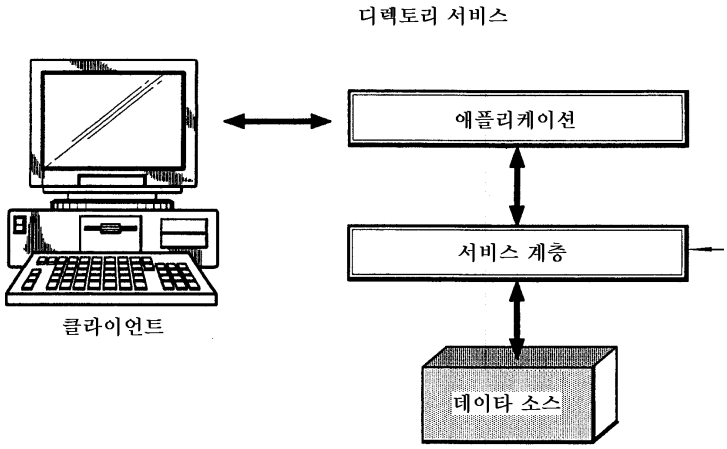
도면2

(관련 기술)

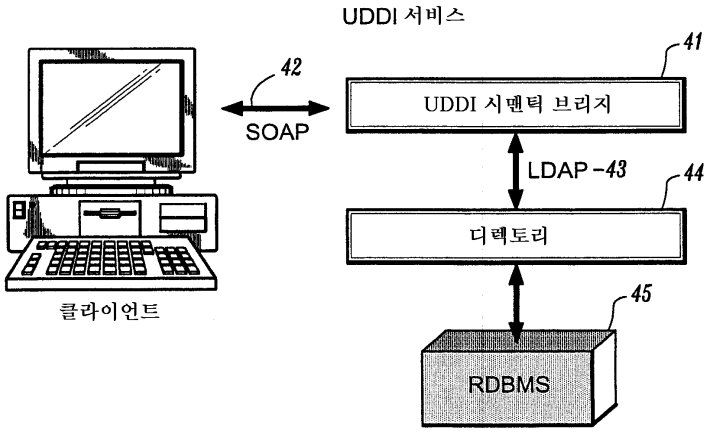


도면3

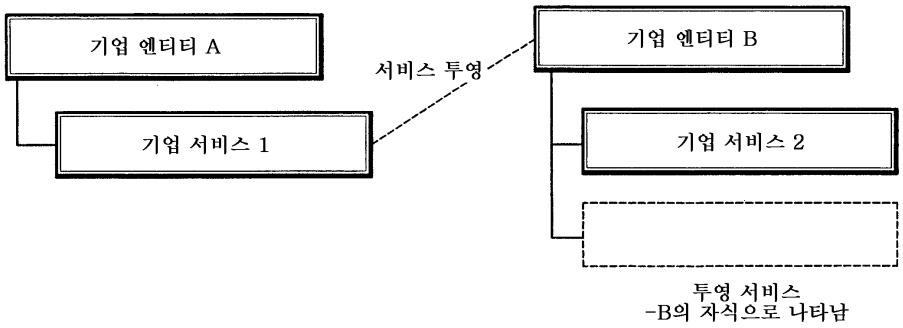
(관련 기술)



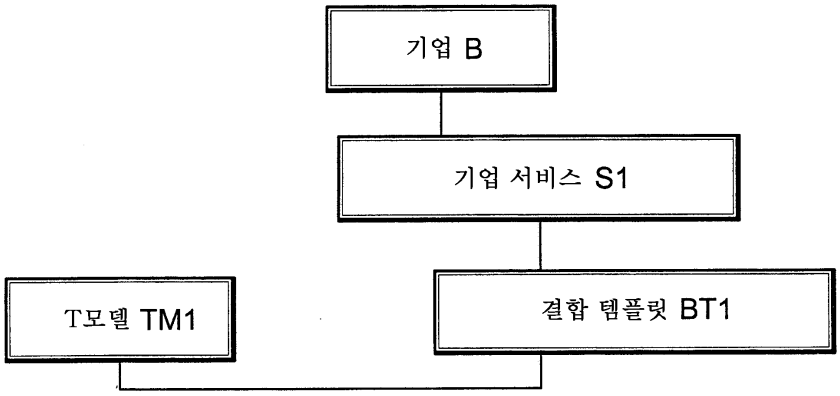

도면4




도면5

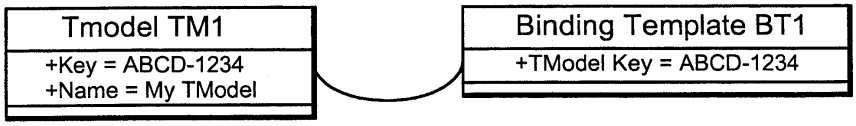


도면6



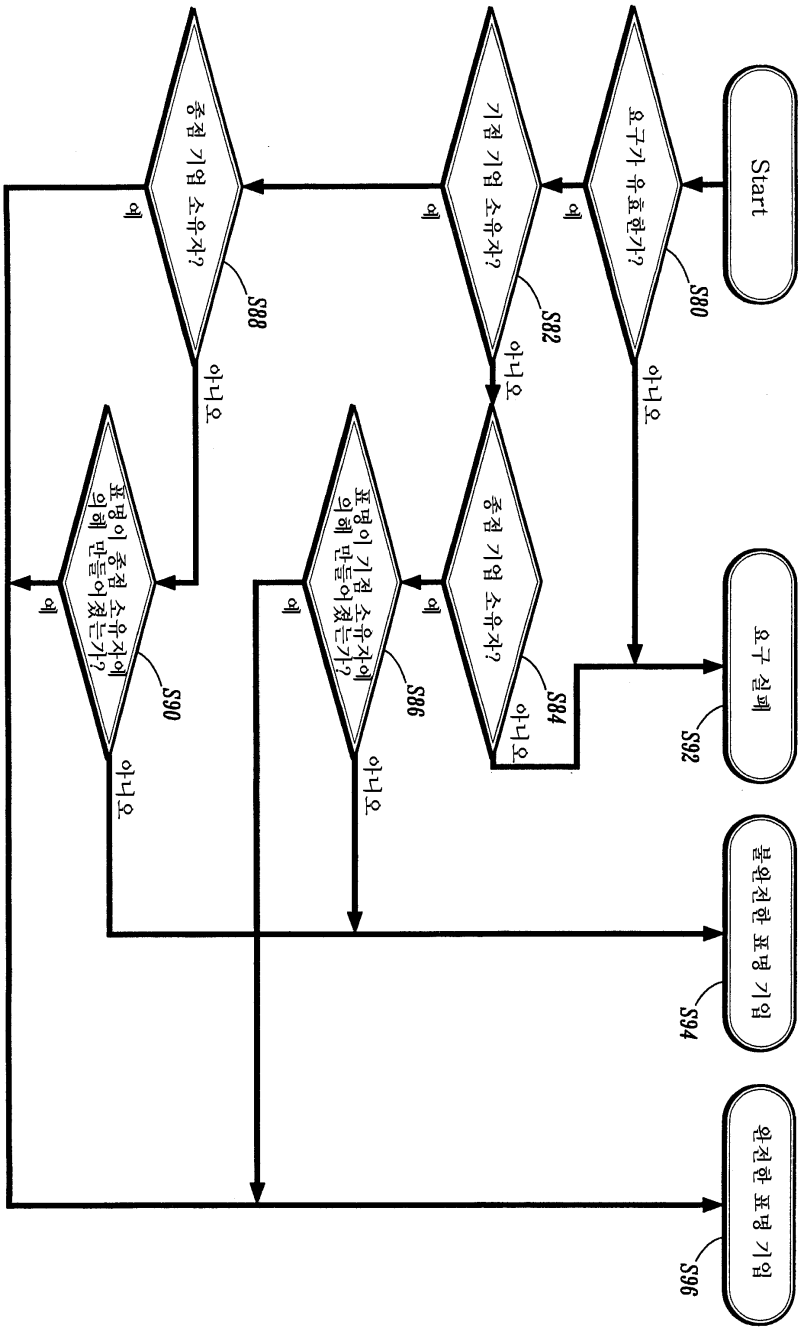

결합 템플릿/T모델 관계

도면7

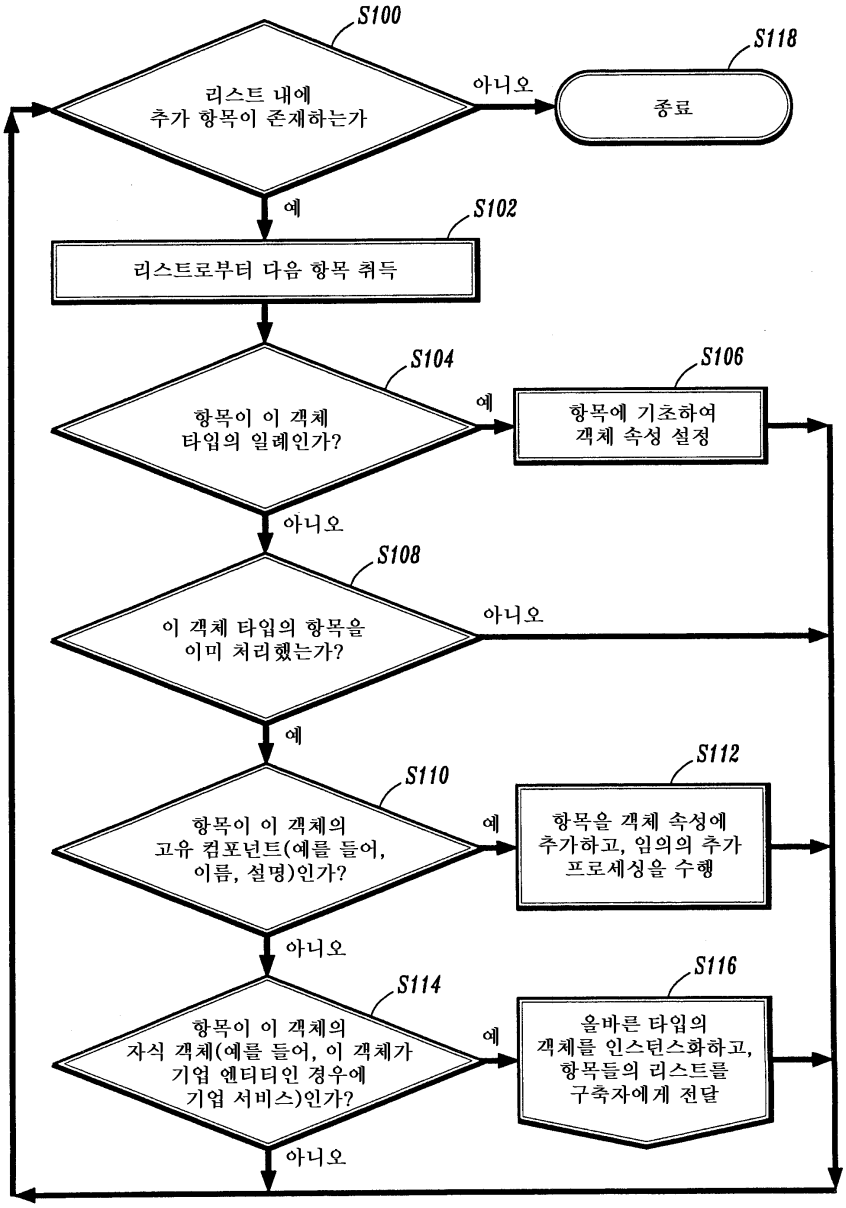



주요 및 외부 키들

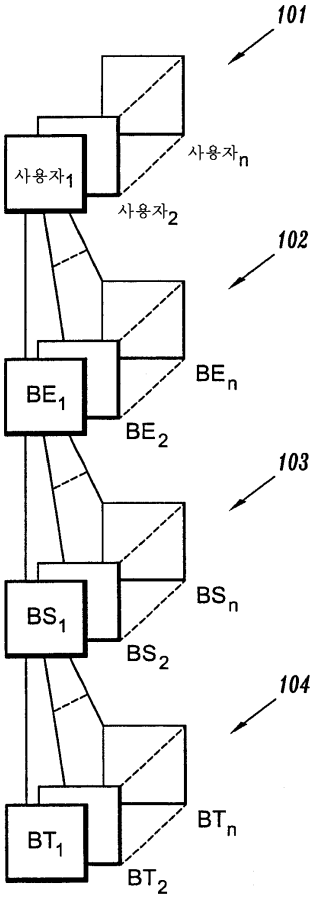
도면8


Start
요구가 유효한가?
S80
예
아니오
기권 기업 소유자?
S82
예
아니오
종결 기업 소유자?
S84
예
아니오
요구 실패
S92
표명이 기권 소유자에 의해 만들어졌는가?
S86
예
아니오
불완전한 표명 기업
S94
종결 기업 소유자?
S88
예
아니오
표명이 종결 소유자에 의해 만들어졌는가?
S90
예
아니오
완전한 표명 기업
S96

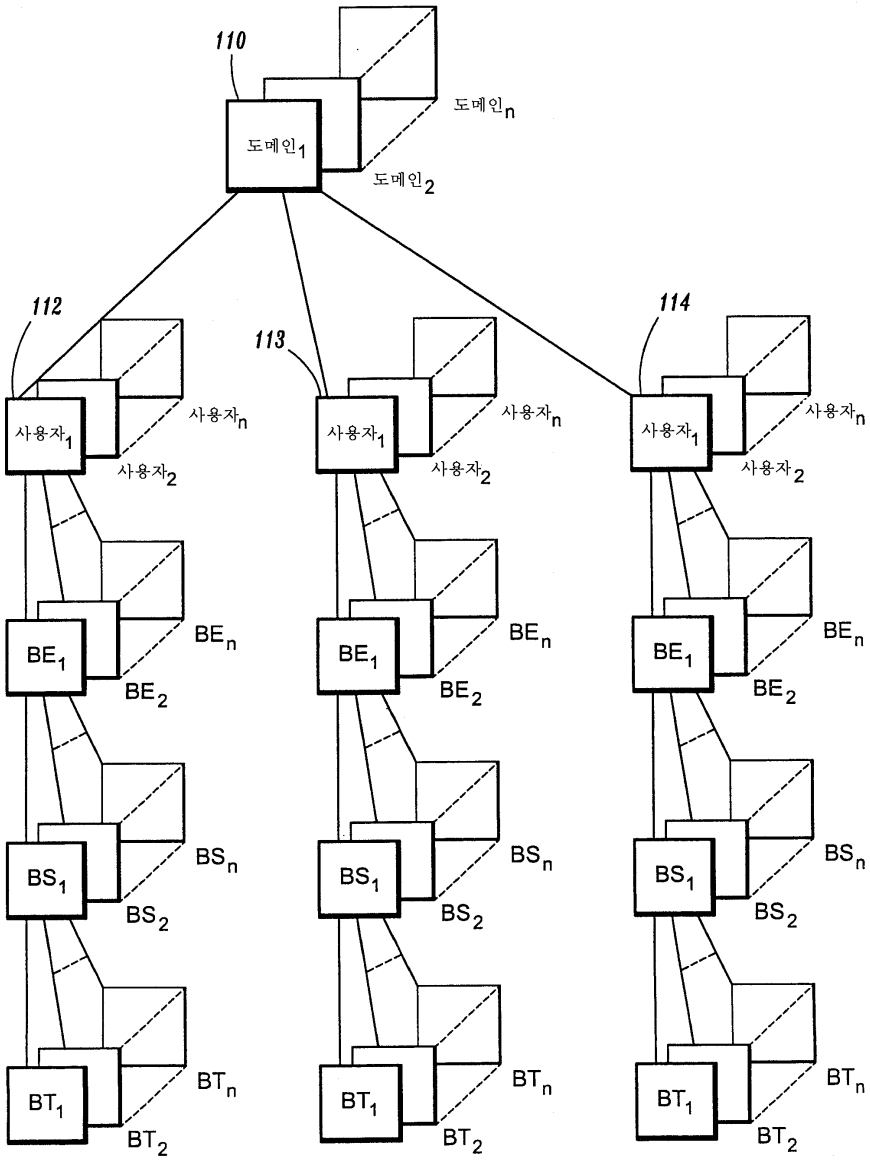
도면9


S100
리스트 내에
추가 항목이 존재하는가
아니오
S118
종료
예
S102
리스트로부터 다음 항목 취득
S104
항목이 이 객체
타입의 일례인가?
예
S106
항목에 기초하여
객체 속성 설정
아니오
S108
이 객체 타입의 항목을
이미 처리했는가?
아니오
예
S110
항목이 이 객체의
고유 컴포넌트(예를 들어,
이름, 설명)인가?
예
S112
항목을 객체 속성에
추가하고, 임의의 추가
프로세싱을 수행
아니오
S114
항목이 이 객체의
자식 객체(예를 들어, 이 객체가
기업 엔티티인 경우에
기업 서비스)인가?
예
S116
올바른 타입의
객체를 인스턴스화하고,
항목들의 리스트를
구축자에게 전달
아니오

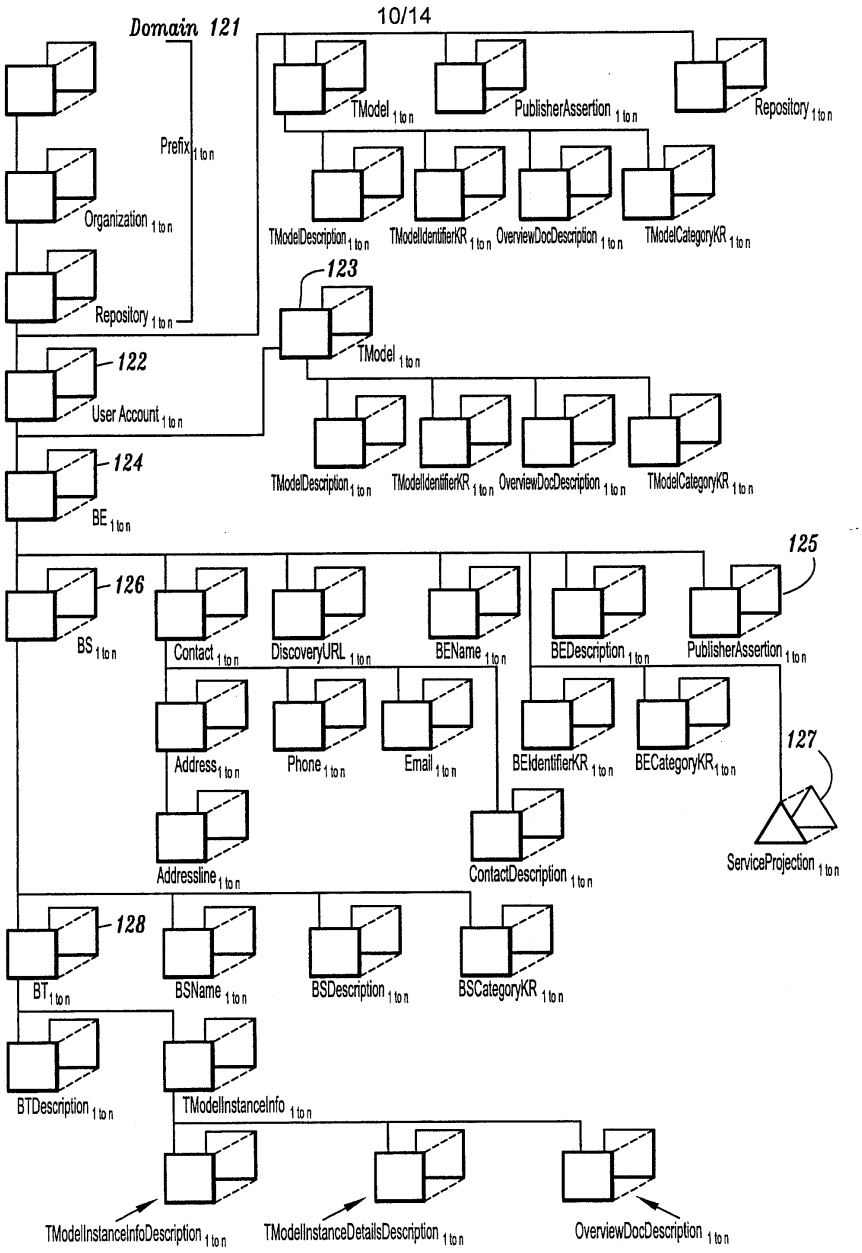
도면10


101
사용자1
사용자n
사용자2
102
BE1
BEn
BE2
103
BS1
BSn
BS2
104
BT1
BTn
BT2

도면11

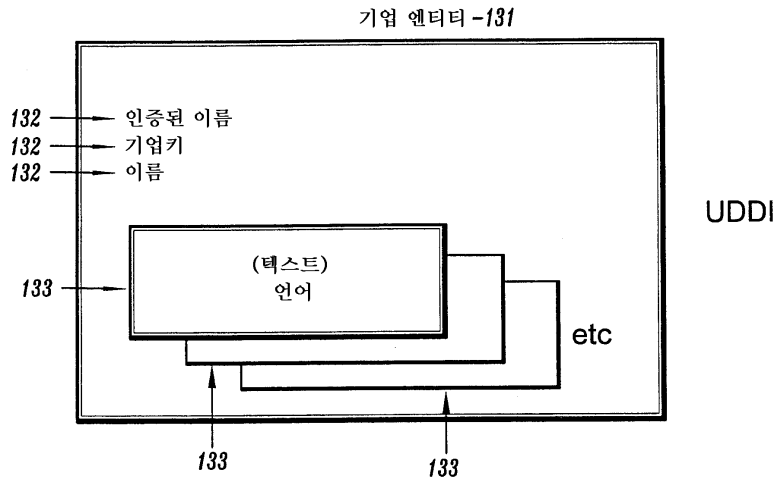

110
도메인1
도메인2
도메인n
112
113
114
사용자1
사용자2
사용자n
BE1
BE2
BEn
BS1
BS2
BSn
BT1
BT2
BTn

도면12


10/14
Domain 121
Prefix 1 to n
Organization 1 to n
Repository 1 to n
TModel 1 to n
PublisherAssertion 1 to n
Repository 1 to n
TModelDescription 1 to n
TModelIdentifierKR 1 to n
OverviewDocDescription 1 to n
TModelCategoryKR 1 to n
123
TModel 1 to n
122
User Account 1 to n
TModelDescription 1 to n
TModelIdentifierKR 1 to n
OverviewDocDescription 1 to n
TModelCategoryKR 1 to n
124
BE 1 to n
125
126
BS 1 to n
Contact 1 to n
DiscoveryURL 1 to n
BEName 1 to n
BEDescription 1 to n
PublisherAssertion 1 to n
Address 1 to n
Phone 1 to n
Email 1 to n
BEIdentifierKR 1 to n
BECategoryKR 1 to n
127
Addressline 1 to n
ContactDescription 1 to n
ServiceProjection 1 to n
128
BT 1 to n
BSName 1 to n
BSDescription 1 to n
BSCategoryKR 1 to n
BTDescription 1 to n
TModelInstanceInfo 1 to n
TModelInstanceInfoDescription 1 to n
TModelInstanceDetailsDescription 1 to n
OverviewDocDescription 1 to n

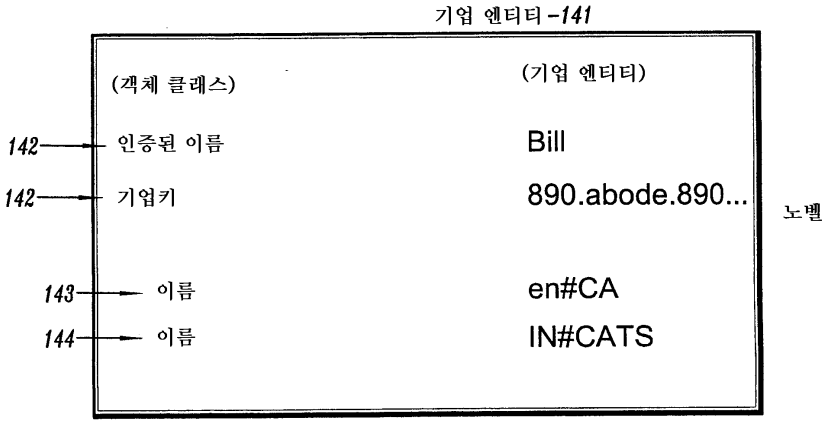
도면13

(종래 기술)

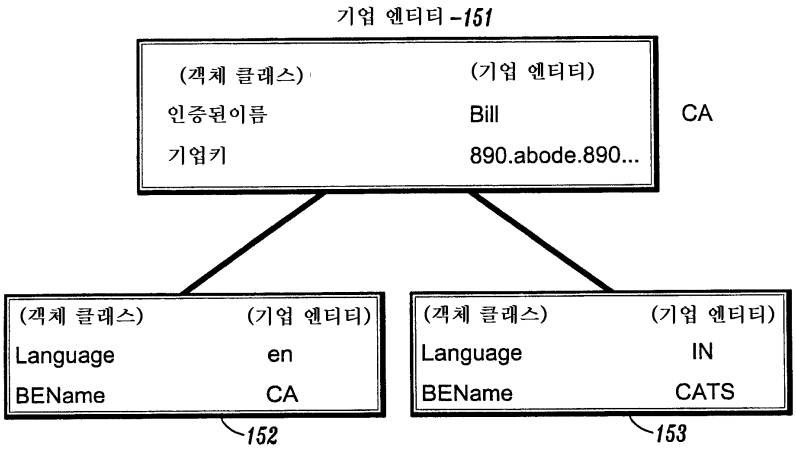



도면14

(종래 기술)

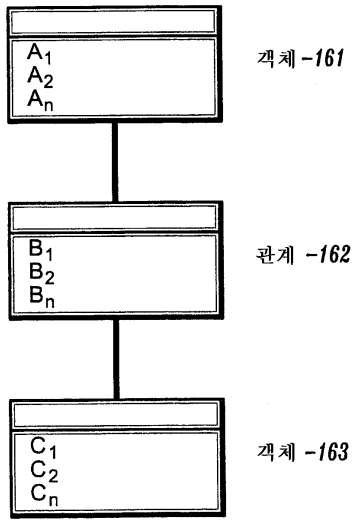



도면15



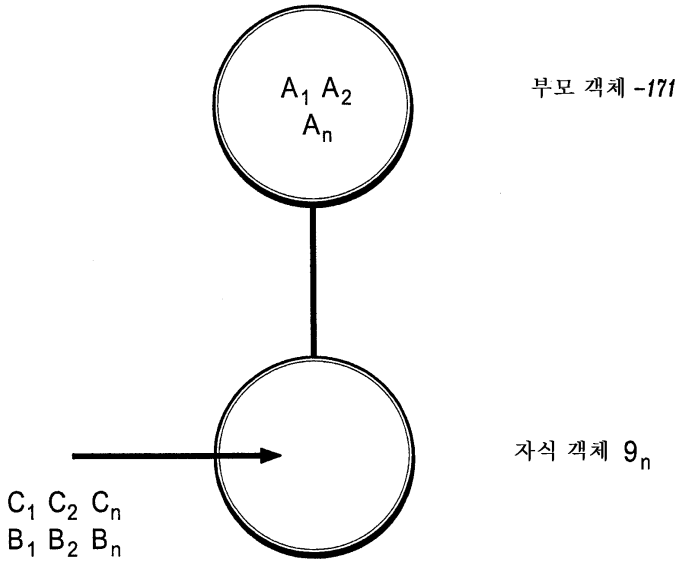
도면16

UDDI 관계도

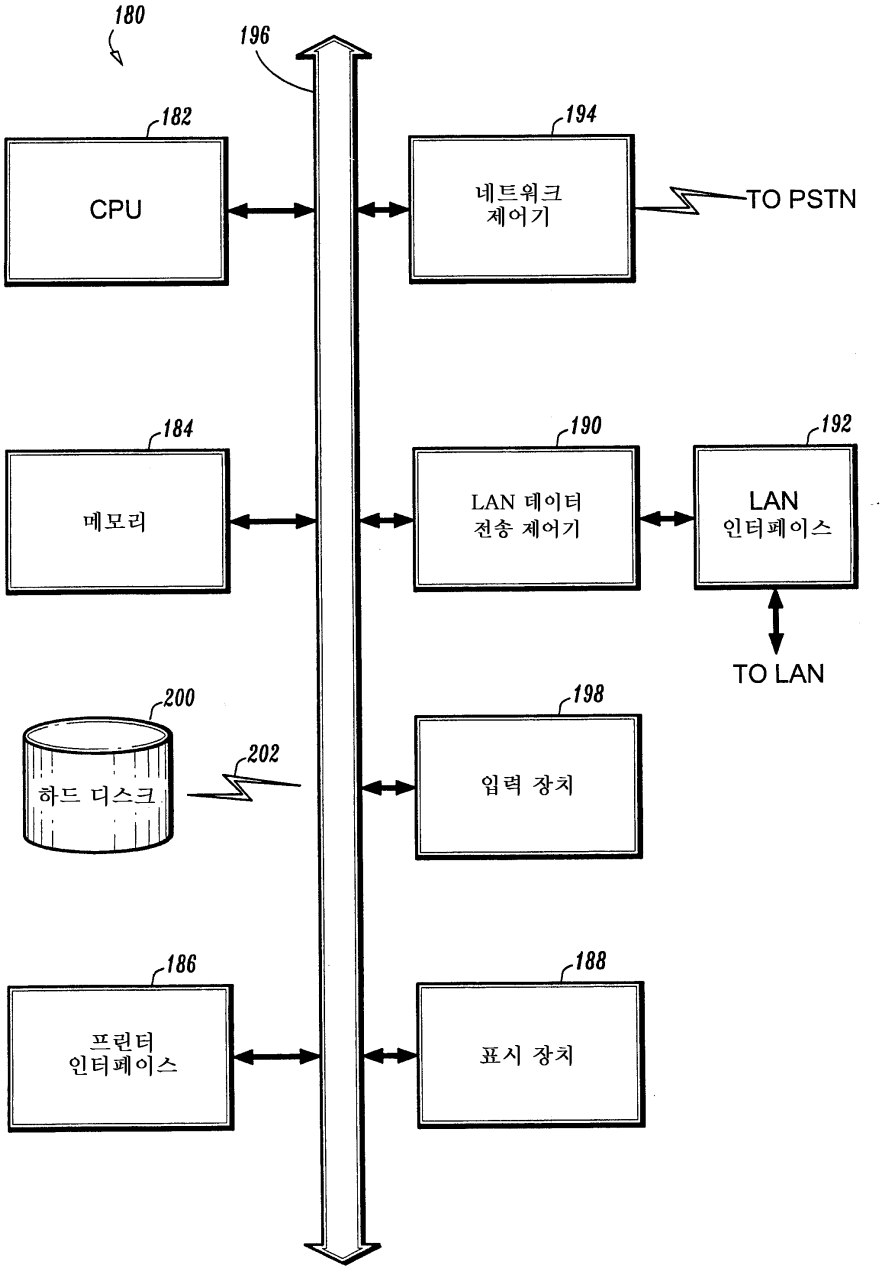



도면17

디렉토리 계층 구조도



도면18


180
196
182
CPU
194
네트워크
제어기
TO PSTN
184
메모리
190
LAN 데이터
전송 제어기
192
LAN
인터페이스
TO LAN
200
하드 디스크
202
198
입력 장치
186
프린터
인터페이스
188
표시 장치