ebXML 레지스트리 기반의 UDDI 서비스 미들웨어 설계 및 구현

(Design and Implementation of a UDDI Service Middleware based on the ebXML Registry)

박 재 홍 † 김 상 균 †† 이 규 철 ††† 김 경 일 ††††
(Jae-Hong Park) (Sang-Kyun Kim) (Kyu-Chul Lee) (Kyung-il Kim)
김 록 원 †††† 송 병 열 †††† 조 현 규 ††††
(Rock-won Kim) (Byoung-youl Song) (Hyun-kyu Cho)

요 약 최근 들어 XML기반의 전자상거래 프레임워크로 ebXML과 웹 서비스가 대두되고 있다. 이들은 사용자 및 어플리케이션들이 정보를 저장하고 검색할 수 있도록 다양한 레지스트리 서비스를 제공하고 있으며, 이를 위해 ebXML은 ebXML 레지스트리를 이용하고 웹 서비스는 UDDI 레지스트리를 이용한다. 이와 같이 ebXML과 웹 서비스는 서로 다른 레지스트리를 사용하고 있지만, 이러한 두 레지스트리의 구조와 기능은 유사한 부분이 많다.

따라서 본 논문에서는 ebXML Registry Information Model(RIM)[1]과 UDDI 데이타구조 사이의 유사성을 분석하여 두 모델간의 매핑 정보를 구성하고 이를 이용하여 UDDI 레지스트리 API[2]를 ebXML 레지스트리 서비스로 변환하는 UDDI 서비스 미들웨어를 설계하고 구현하였다. 이 시스템을 이용하면 ebXML 레지스트리에 별도의 변경을 가하지 않고도 ebXML 레지스트리를 ebXML 레지스트리뿐만 아니라 UDDI 레지스트리로도 사용할 수 있기 때문에 e-비즈니스를 하기 위해 두 개의 레지스트리를 모두 도입할 필요가 없는 장점이 있다.

키워드 : ebXML, 웹서비스, e-비즈니스

Abstract Recently, ebXML and Web Services are emerging as the XML-based electronic business frameworks. To provide a set of registry services which users or applications can store and search the business information, ebXML and Web Services use ebXML registry and UDDI registry, respectively. They use the different registries each other, but there are something that similiar to the structure and functions of both of registries.

In this paper, we analyze the similarity of ebXML Registry Information Model(RIM) and UDDI data structure, and construct the mapping information. So, we design and implement the UDDI Service Middleware which translates the request of UDDI registry service into the request of ebXML registry service using the algorithm. Consequently, we could use the ebXML registry like ebXML registry as well as UDDI registry without any changes in the ebXML registry itself so that we can not need to have both of registries for e-business

Key words : ebXML, Web Services, e-Business

† 비 회 원 : KT 융용시스템연구소
jhpark@ce.cnu.ac.kr
†† 학생회원 : 충남대학교 컴퓨터공학과
skkim@ce.cnu.ac.kr
††† 종신회원 : 충남대학교 컴퓨터공학과 교수
kclee@ce.cnu.ac.kr
†††† 비 회 원 : 한국전자통신연구원 지능형서비스플랫폼연구팀 연구원
zbmon@etri.re.kr
rwkim@etri.re.kr
sby@etri.re.kr
hkcho@etri.re.kr
논문접수 : 2003년 4월 21일
심사완료 : 2004년 2월 4일

1. 서 론

최근 들어, 각 표준 단체나 기업들은 인터넷 기반의 e-비즈니스를 위한 다양한 프레임워크를 개발하고 있으며, 제안된 프레임워크들은 각 기업들의 비즈니스에 관

1) 레지스트리의 정보 모델을 표현하기 위해서 UDDI 레지스트리에서는 데이타구조라는 용어를 사용하고 ebXML 레지스트리에서는 RIM이라는 용어를 사용한다.

2) 레지스트리에서 제공하는 서비스를 나타내기 위해서 UDDI 레지스트리에서는 API라는 용어를 사용하고 ebXML 레지스트리에서는 레지스트리 서비스라는 용어를 사용한다.

련된 기업정보 및 서비스 정보등을 레지스트리에 저장하고 여러 사용자 및 어플리케이션들이 레지스트리에 저장된 정보를 쉽게 공유할 수 있도록 다양한 레지스트리 서비스를 제공하고 있다. 현재까지 많은 프레임워크들이 제안되었지만 그 중에서도 ebXML(electronic business eXtensible Markup Language)[1]과 UDDI (Universal Description, Discovery and Integration)[2]로 대표되는 웹 서비스(Web Services)가 기업들의 관심대상이 되고 있다. ebXML은 "Creating A Single Global Market"이라는 기치 아래 그 동안 국제 EDI 표준을 추진해 왔던 UN/CEFACT과 OASIS가 주축이 되어, XML을 이용하여 인터넷 기반의 e-비즈니스가 가능하도록 제정한 전자 상거래 표준이다. ebXML 표준은 2001년 5월에 완성되었으며, 그 후에도 웹 서비스와 같은 다양한 표준을 ebXML에서 수용하는 등 표준의 가치를 높이기 위한 연구가 진행중이다. 특히 ebXML에서 가장 중요한 역할을 하는 ebXML 레지스트리에 대한 표준은 현재 ebXML RIM(Registry Information Model)[3]과 RS(Registry Service)[4] 모두 버전 2.1이 2002년 6월에 발표되었다. ebXML 레지스트리 2.x 스펙은 기존의 버전 1.0을 수용하면서 웹 서비스의 기능을 수행할 수 있도록 만들어졌다. 이를 위해 ebXML 레지스트리 모델에 서비스와 관련된 클래스를 추가하였으며, 거래 상대자들 사이에 메시지를 교환할 때 ebXML MS(Messaging Service)[5]만을 이용하는 것이 아니라 SOAP(Simple Object Access Protocol)[6]을 이용한 방법도 지원하고 있다.

웹 서비스는 표준화된 XML 메시지를 통해 네트워크 상에서 접근 가능한 연산들의 집합을 기술하는 인터페이스이다. 또한 현재의 XML 기술을 기반으로 기존의 웹 환경을 이용한 분산 컴퓨팅을 가능케 함으로써 웹을 통한 시스템 통합을 용이하게 한다. 웹 서비스는 UDDI 레지스트리를 기반으로 웹 서비스의 등록, 검색을 위한 기반 기술을 제공하며, 웹 서비스의 요청 및 응답에서 사용되어지는 메시지 형식 위해 SOAP을 정의하고, 웹 서비스 접근에 이용되는 SOAP 메시지, 프로토콜, 웹 서비스에 대한 인터넷 위치 정보 등을 기술하기 위해 WSDL(Web Service Description Language)[7]을 지원한다.

실제 기업들이 이와 같은 ebXML과 UDDI로 대표되는 웹 서비스를 이용하여 비즈니스를 수행하기 위해서는 두 표준 각각에 대한 솔루션을 모두 가지고 있어야 하는데 이를 위해서 필요한 비용과 노력 때문에 두 가지 표준을 모두 도입한다는 것은 힘든 상황이다. 그러나 현재 웹 서비스는 Microsoft, IBM, Ariba를 포함한 기업과 단체가 주축이 되어 UDDI 레지스트리를 구축하였고, 웹 서비스에 대한 솔루션이 ebXML보다 많이 개발되어 있으며, 웹 서비스를 도입하는 비용도 저렴하기 때문에 대부분 e-비즈니스를 하려는 기업에서는 ebXML보다 웹 서비스를 많이 이용하고 있다[8,9]

하지만 ebXML은 웹 서비스의 기능을 수용하는 등 일반적이고 표준화된 프레임워크로 만들어졌기 때문에 앞으로 ebXML에 대한 솔루션 개발이 활발해지면 웹 서비스보다 e-비즈니스에 적합한 프레임워크가 될 수 있다.

이와 같이 현재는 UDDI로 대표되는 웹 서비스에 대한 이용이 증가하고 있지만 ebXML은 웹 서비스 보다 일반적인 프레임워크를 제공하는 등 서로의 장단점이 있기 때문에 e-비즈니스를 하기 위해서 두 솔루션을 모두 도입해야 한다. 하지만 위에서 언급한 바와 같이 비용, 노력과 같은 문제 때문에 현실적으로는 어렵다.

따라서 본 연구에서는 UDDI 레지스트리보다 일반적인 구조와 기능을 가지는 ebXML 레지스트리에서 ebXML 레지스트리뿐만 아니라 UDDI 레지스트리의 기능까지 이용할 수 있도록 하는 UDDI 서비스 미들웨어를 설계하고 구현하였다. 이를 위해서 우선 ebXML RIM과 UDDI의 데이타구조 사이의 유사성을 분석하여 두 구조 사이의 매핑 알고리즘을 개발하고, 이 알고리즘을 이용해서 UDDI 레지스트리 서비스 요구를 ebXML 레지스트리 서비스 요구로 변환해줄 수 있도록 하였다.

만약 기업에서 본 연구에서 구현한 미들웨어를 도입한다면 두 프레임워크를 모두 도입하지 않고도 ebXML 레지스트리에서 ebXML 레지스트리 서비스뿐만 아니라 UDDI API를 처리할 수 있기 때문에 ebXML의 주 목적인 B2B collaboration을 지원함과 동시에 웹 서비스의 주 목적인 Application Integration 지원을 가능하게 할 수 있다. 또한 하나의 기업에서 ebXML의 도입만으로도 웹 서비스를 동시에 도입할 수 있는 비용절감 효과를 기대할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 전체 시스템 구조를 설명하고, 3장에서는 ebXML RIM과 UDDI의 데이타구조의 유사성을 분석하여 매핑 알고리즘을 제시한다. 4장에서 이 알고리즘을 이용한 UDDI 서비스 미들웨어의 동작 메커니즘에 대해 설명하고, 5장에서는 관련 연구에 대해서 소개한다. 그리고 마지막 5장에서는 결론과 향후 연구방향에 대하여 논한다.

2. 시스템 구조

2.1 시스템 구조

그림 1은 본 논문에서 개발한 UDDI 서비스 미들웨어와 이 시스템이 동작하는 전체 시스템 구조를 나타낸

것이다. 본 논문에서 구현한 UDDI 서비스 미들웨어는 UDDI 클라이언트[10]와 ebXML 레지스트리[11-13] 사이에서 UDDI 클라이언트의 서비스 요구를 ebXML 레지스트리 서비스로 변환해 준다. 이러한 변환을 통해 ebXML 레지스트리 상에서 UDDI 레지스트리 서비스를 처리함으로써 ebXML 레지스트리를 UDDI 레지스트리처럼 사용할 수 있게 한다.

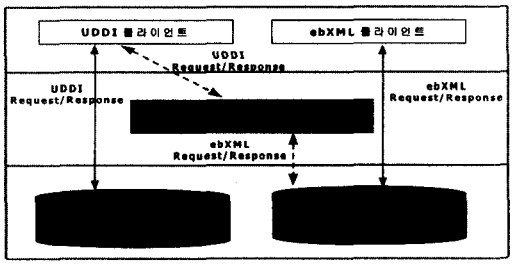

그림 1 전체 시스템 구조

2.2 UDDI 서비스 미들웨어

그림 2는 UDDI 서비스 미들웨어의 세부 구조를 나타낸 것이다. UDDI 서비스 미들웨어는 요구메시지 분석기, 결과메시지 작성기, UDDI to ebXML Request 매핑모듈, ebXML to UDDI Response 매핑모듈로 이루어져 있다.

요구메시지 분석기는 UDDI 클라이언트로부터 전달받은 메시지를 파싱해서 유효성을 검사한 후에 UDDI API를 추출하여, UDDI to ebXML Request 매핑모듈로 전달한다. 이 매핑 모듈은 요구메시지 분석기로부터 전달받은 UDDI API를 데이타구조 매핑테이블을 참고하여 ebXML 서비스 요구로 변환한 후 ebXML 레지스트리 서버에게 서비스를 요청한다. ebXML 레지스트리 서버는 이 서비스를 처리한 후에 응답 메시지를 ebXML to UDDI Response 매핑모듈로 보낸다. ebXML to UDDI Response 매핑모듈은 데이타구조 매핑테이블을 이용하여 ebXML Response를 UDDI Response로 변환한 후, 변환된 결과를 결과메시지 작성기에게 전달하게 되고, 결과메시지 작성기는 전달받은 UDDI Response에 SOAP 헤더를 붙여 UDDI 클라이언트에게 최종적으로 전달하게 된다.

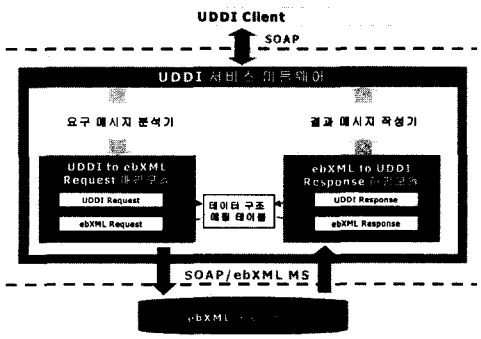

그림 2 UDDI 서비스 미들웨어의 시스템 구조

3. ebXML-UDDI 데이타구조 매핑

3.1 ebXML RIM과 UDDI 데이타구조

그림 3은 ebXML 레지스트리 버전 2.1 RIM에 대한 High-Level 클래스 다이어그램으로서, 크게 ebXML 레지스트리를 사용하는 단체와 사용자에 대한 정보를 나타내는 Organization과 User, 레지스트리에서 제공하는 서비스에 대한 정보인 Service, 객체들의 분류에 대한 정보를 가지고 있는 Classification 그리고 모든 레지스트리에 저장된 정보를 나타내는 RegistryObject로 이루어져 있다.

그림 4는 UDDI 데이타구조의 High-Level 클래스 다이어그램을 나타내고 있다. UDDI의 데이타구조는 그림과 같이 크게 5가지 요소들로 이루어져 있으며 각 요소에 대한 설명은 그림에 나와 있다.

3.2 데이타구조 매핑

ebXML RIM과 UDDI 데이타구조는 각 시스템의 정보를 모델링한 것으로 표현하는 방법과 범위에 서로 조금씩 차이가 있지만 모두 e-비즈니스에 필요한 정보를 표현할 수 있다. 따라서 본 연구에서는 UDDI API를 ebXML 서비스로 변환하기 위해서 우선 두 시스템의 정보 모델에 대한 의미를 분석하고 비슷한 의미를 가지는 클래스와 클래스의 애트리뷰트들간의 매핑 테이블을 구성한다. 이 경우에 UDDI API를 ebXML 서비스 요구로 변환하기만 하면 되기 때문에 UDDI API의 정보가 ebXML 서비스에서 어떤 정보에 매핑되는지를 분석하며 반대의 경우 즉, UDDI API 구성에 필요 없는 ebXML 서비스 정보가 UDDI API에 어떻게 매핑되는지에 대해서는 고려하지 않는다. 또한 이와 같은 매핑 작업은 두 모델의 의미를 분석해야 하는 이유로 어떤 규칙을 적용하기 힘들기 때문에 자동화된 방법을 사용하지 않고 사람 손으로 직접 매핑하였다.

그림 5는 두 정보 모델에서 가장 기본적인 클래스들 간의 매핑을 개념적으로 표현한 그림으로써, 변환 대상이 UDDI API이기 때문에 그림 5는 UDDI의 클래스에 대해서 ebXML의 어떤 클래스들과 연관될 수 있는지를 보이고 있다. 하지만 그림 5의 모든 클래스는 정확히 일대일로 매핑되지 않는다. 예를 들어 UDDI의 businessService 클래스에 존재하는 모든 애트리뷰트가 오직 ebXML의 Service클래스의 애트리뷰트들과 매핑되지 않는다.

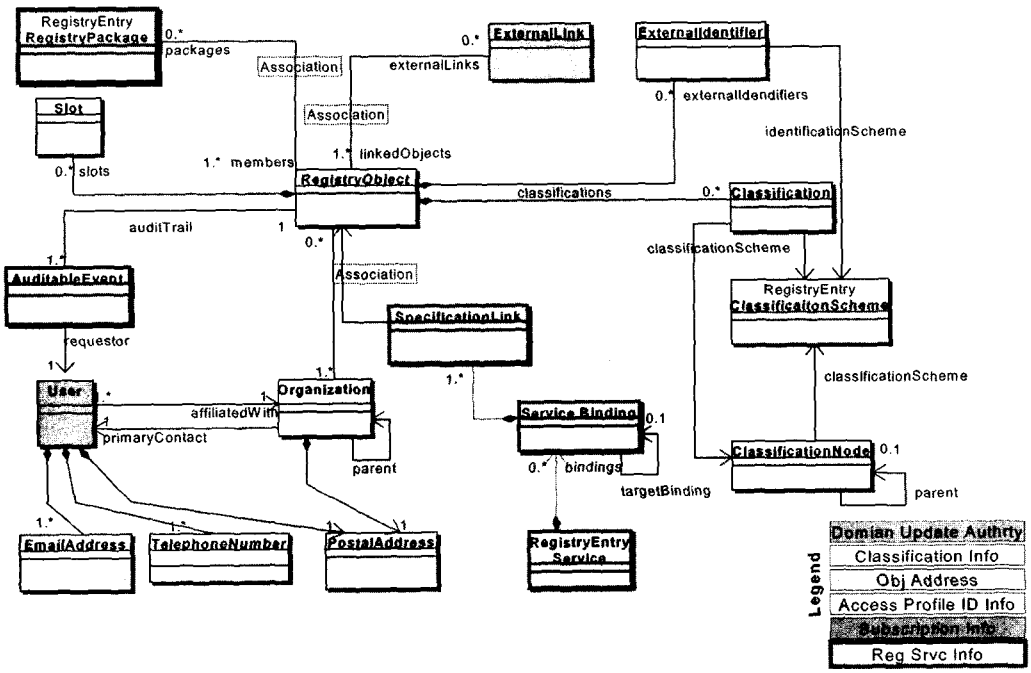

그림 3 ebXML 버전 2.1 RIM의 High-Level 클래스 다이어그램

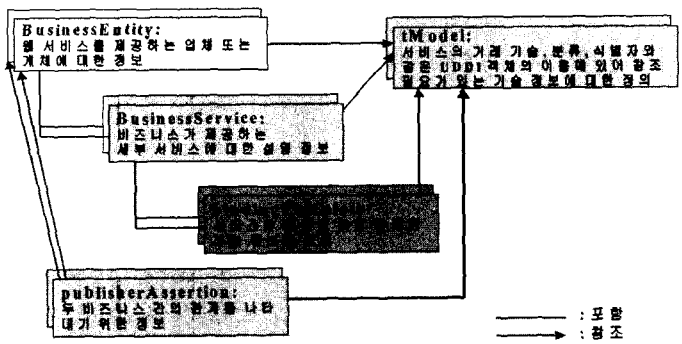

그림 4 UDDI 버전 2.0 데이타구조의 High-Level 클래스 다이어그램

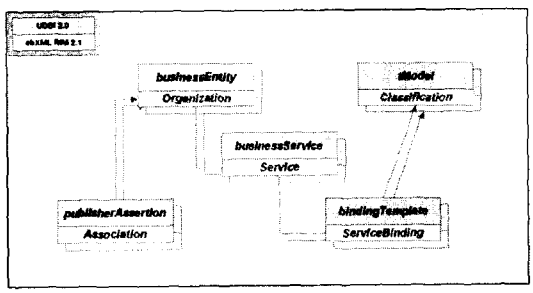

그림 5 UDDI 버전 2.0과 ebXML RIM 버전 2.1간의 클래스 매핑 관계

그림 6은 UDDI에서 businessService 클래스의 각 애트리뷰트가 ebXML의 어떤 애트리뷰트들과 매핑되는지에 대한 그림이다. 그림에서 'v' 표시는 꼭 데이타가 저장되어 있어야 하는 필수 필드를 나타내며, businessServices클래스는 기본 데이타 타입을 가지는 필드 외에도 bindingTemplate과 CategoryBag 서브 클래스를 가지는데 이들은 ebXML에서 각각 ServiceBinding과 Classification 클래스에 매핑된다. businessService 클래스 외에도 모든 UDDI 클래스와 클래스의 애트리뷰트들에 대해서 ebXML RIM과의 매핑을 이미 [14][15]에서 수행하였으며 따라서 본 논문에서는 모든 매핑 정

UDDI - businessService				ebXML - Service			
	Field Name	Type	Length		Field Name	Type	Length
	businessKey	UUID	41	v	Organization.id	string	64
v	serviceKey	UUDI	41	v	Service.id	string	64
v	name	string	255		Name.value	string	256
	description	string	255		Description.value	string	256
v	bindingTamplate	struct			ServiceBinding		
	categoryBag	struct			Classification		

그림 6 businessService의 정보구조 매핑 테이블

보에 대해서는 언급하지 않는다.

ebXML RIM과 UDDI 데이타구조를 매핑하는 과정에서 그림 6에서도 볼 수 있듯이 서로의 데이타 구조가 완전히 일치하지는 않는 것을 발견할 수 있다. 예를 들면, businessService클래스의 name애트리뷰트는 필수 항목이지만 이와 매핑되는 Name.value는 옵션 항목이며, name 애트리뷰트의 크기는 255바이트이지만 Name.value 애트리뷰트는 256바이트를 가지고 있다. 이러한 데이타구조 매핑시의 불일치 문제는 다음 절에서 자세히 설명한다.

3.3 데이타구조 불일치 분류 및 해결방법

이 장에서 분류한 문제들은 본 연구에서 구현하는 UDDI 서비스 미들웨어에서 발생하는 문제들로써, 이 문제들에 대한 해결 방법은 UDDI 서비스 미들웨어 또는 ebXML 서버에서 가능한 것들을 제시하며, UDDI 또는 ebXML 클라이언트에서의 해결방안은 고려하지 않는다.

3.3.1 필수 제약조건 불일치

UDDI 데이타구조에서 필수항목과 ebXML RIM에서 선택항목을 서로 매핑할 경우, 또는 그 반대의 경우에 발생하는 문제이며, 다음의 두 경우가 존재한다.

가. UDDI 데이타구조에서 필수항목과 ebXML RIM에서 선택항목을 매핑하는 경우

UDDI 데이타구조에서 필수항목을 ebXML RIM의 선택항목과 매핑할 경우, ebXML 클라이언트가 Publishing을 수행할 때 선택항목을 저장하지 않으면 UDDI 검색 결과 중에서 필수항목의 값이 존재하지 않게 되는 문제가 발생한다. 이 문제를 해결하기 위해서는 ebXML 레지스트리의 스키마에서 제약조건을 필수로 바꾸어야 하지만 본 시스템에서는 서버의 스키마 제약조건을 바꿀 수 없기 때문에 불가능하다.

나. UDDI 데이타구조에서 선택항목과 ebXML RIM에서 필수항목을 매핑하는 경우

UDDI 데이타구조에서 선택항목을 ebXML RIM의 필수항목과 매핑하는 경우, UDDI 서비스 미들웨어에서 Publishing을 수행할 때 선택항목에 값이 존재하지 않으면 ebXML 레지스트리 스키마의 필수 제약조건 때문에 저장할 수가 없다. 이러한 경우에는 가상의 데이타를 추가하여 저장함으로써 불일치를 해결할 수 있다. 이렇게 하면 UDDI 클라이언트는 검색결과에 가상데이타가 포함되지 않기 때문에 상관없지만 ebXML 클라이언트에서 이 데이타를 검색하면 의미 없는 결과가 나올 수 있다.

3.3.2 데이타 크기 제약조건 불일치

UDDI 데이타구조와 ebXML RIM의 스키마에서 매핑되는 항목들의 최대 허용 데이타 크기가 서로 다른 경우 발생하는 문제이며 다음의 두 경우가 존재한다.

가. UDDI 데이타구조에서의 허용 크기가 ebXML RIM에서의 허용 크기보다 작은 경우

매핑되는 두 개의 항목 간에 UDDI 데이타구조에서 허용하는 데이타 크기가 ebXML RIM에서 허용하는 데이타 크기보다 작은 경우 ebXML 클라이언트가 저장한 데이타에 대해서 UDDI 서비스 미들웨어가 Inquiry를 수행할 때 검색 결과의 끝이 잘려서 일부만 리턴될 수

UDDI - BusinessEntity				ebXML - Organization			
Field Name	Example	Type	Length	Field Name	Example	Type	Length
v Name	'DB Lab.'	string	255	Name	'DB Lab.'	string	256

그림 7 UDDI 데이타구조에서는 필수항목이고 ebXML RIM에서는 선택항목인 경우의 예

UDDI - Contacts				ebXML - User			
Field Name	Example	Type	Length	Field Name	Example	Type	Length
email	jhpark@ce.cnu.ac.kr	string	255	v email	jhpark@ce.cnu.ac.kr	string	128

그림 8 UDDI 데이타구조에서는 선택항목이고 ebXML RIM에서는 필수항목인 경우의 예

있다. 이 문제를 해결하기 위해서는 ebXML RIM의 스키마에서 최대 허용 데이타 크기를 줄여줘야 하지만 3.2.1절의 1번 문제와 마찬가지로 현재는 불가능하다.

나. UDDI 데이타구조에서의 허용 크기가 ebXML RIM에서의 허용 크기보다 큰 경우

UDDI 데이타구조에서 허용하는 데이타 크기가 ebXML RIM에서 허용하는 데이타 크기보다 큰 경우 UDDI 레지스트리에서 Publishing을 수행할 때 데이타의 일부만 저장될 수 있다. 이 경우 데이타를 ebXML RIM의 Slot클래스를 이용하여 분할저장하며, 이 후에 UDDI 레지스트리에서 이 데이타를 Inquiry하는 경우 Slot에 나누어 저장된 내용을 합병하여 돌려줌으로써 불일치를 해결할 수 있다. 하지만 ebXML클라이언트에서는 허용 크기가 작기 때문에 Slot의 내용을 합병한다고 하더라도 데이타가 잘리게 된다. 따라서 ebXML에서 Inquiry를 수행하는 경우에는 이 문제를 해결할 수 없다.

3.3.3 충돌 문제 해결에 따른 시스템 사용

본 연구에서 개발한 UDDI 서비스 미들웨어를 이용하면 UDDI 클라이언트는 ebXML 레지스트리를 마치 UDDI 레지스트리처럼 이용할 수 있다. 또한 UDDI 클라이언트와 ebXML 클라이언트 모두 ebXML 레지스트리에 Publishing과 Inquiry를 수행할 수 있게 된다. 하지만 이 경우 3.2.1절과 3.2.2절에서 설명한 바와 같이 여러 가지 불일치 문제가 발생한다. 그림 11은 ebXML 레지스트리에 UDDI 또는 ebXML 클라이언트가 Publishing하고 Inquiry할 때 발생하는 데이타 구조 불일치 문제와 해결 여부를 나타낸 것이다.

ebXML 클라이언트에서 Publishing한 정보를 UDDI 서비스 미들웨어에서 Inquiry를 수행하는 경우 3.2.1절의 1번과 3.2.2절의 1번에서와 같이 UDDI 서비스 미들웨어에서 Inquiry를 수행할 때 문제가 발생하지만 이

Publishing	Inquiry	데이터 구조 불일치	해결 여부
UDDI	UDDI	UDDI Publishing과 Inquiry 수행시 3.2.1절의 2번과 3.2.2절의 2번 불일치	해결 가능
ebXML	UDDI	UDDI Inquiry 수행시 3.2.1절 1번과 3.2.2절 1번 불일치	해결 불가능
UDDI	ebXML	ebXML Inquiry 수행시 3.2.1절 2번과 3.2.2절 2번 불일치	해결 불가능
ebXML	ebXML	불일치 없음	필요 없음

그림 11 충돌 문제 해결에 따른 시스템 사용

불일치는 해결할 수 없다.

반대의 경우에도 즉, UDDI 서비스 미들웨어에서 Publishing한 정보를 ebXML 클라이언트에서 Inquiry를 수행하는 경우 3.2.1절의 2번과 3.2.2절의 2번과 같은 불일치가 발생한다. 이 경우에는 UDDI 서비스 미들웨어에서는 Publishing할 때 가상데이타를 추가하거나 Slot에 분할 저장하고 Inquiry할 때 Slot을 이용한 합병을 이용하면 문제를 해결할 수 있다. 하지만 ebXML 레지스트리에서는 Inquiry 수행시 의미 없는 데이타가 검색되거나 데이타의 내용이 잘릴 수 있기 때문에 문제가 발생하게 된다.

하지만 UDDI 클라이언트에서 본 UDDI 서비스 미들웨어를 이용해서 ebXML 레지스트리에 Publishing한 정보를 UDDI 클라이언트에서만 Inquiry하는 경우 3.2.1절의 2번과 3.2.2절의 2번과 같은 불일치가 발생하지만, 각각에서 제시한 불일치 해결방법인 가상데이타 추가, 분할저장과 합병을 통하여 UDDI API의 데이타 무결성을 보장할 수 있다. 따라서 본 연구에서 설계하고 구현하는 시스템은 이 경우만을 고려한다.

4. UDDI 서비스 미들웨어의 설계 및 구현

본 장에서는 ebXML레지스트리에서 UDDI API를 처

	UDDI - BusinessEntity				ebXML - Organization			
	Field Name	Example	Type	Length	Field Name	Example	Type	Length
v	Name	'DB Lab.'	string	255	Name	'DB Lab.'	string	256

그림 9 UDDI 데이타구조에서의 허용 크기가 ebXML RIM에서의 허용 크기보다 작은 경우의 예

	UDDI - Contacts				ebXML - User			
	Field Name	Example	Type	Length	Field Name	Example	Type	Length
v	personName	'Park Jaehong'	string	255	personName_firstName personName_middleName personName_LastName	Park Jaehong	string	64 64 64
					Slot.value	추가입력내용	string	64

그림 10 UDDI 데이타구조에서의 허용 크기가 ebXML RIM에서의 허용 크기보다 큰 경우의 예

리할 수 있는 UDDI 서비스 미들웨어의 동작 알고리즘 설계와 구현에 대해 설명한다. 그림 12는 UDDI 레지스트리에서 지원해야 하는 UDDI API들로써 16개의 Publishing API와 10개의 Inquiry API로 구성되며, 그림 13은 ebXML에서 지원하는 서비스들로써 본 논문에서 구현한 UDDI 서비스 미들웨어는 그림 12의 모든 UDDI API를 적당한 그림 13의 ebXML 서비스로 변환하게 된다.

이러한 변환을 위해서 본 연구에서는 UDDI API를 ebXML 서비스로 변환하는 알고리즘을 설계 구현하였으며 이 알고리즘들의 종류와 기능은 표 1에 나와 있다. 각각의 변환 알고리즘은 UDDI API의 구조를 분석하고 UDDI API에 나타난 엘리먼트를 [14][15]에서 보인 데이타구조 매핑에 따라서 일대일로 변환하여 최종적으로

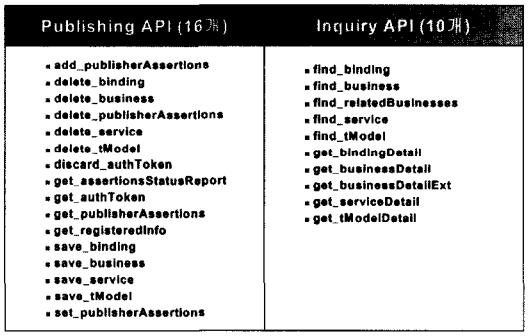

그림 12 UDDI API

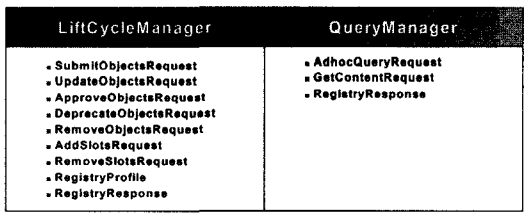

그림 13 ebXML 서비스

ebXML 서비스를 생성하는 작업을 수행한다.

위의 변환 알고리즘은 2장에서 기술한 시스템 구조하에서 실행되며 이를 위해 다음과 같은 구현 환경을 구축하였다.

- 개발 시스템 환경
  - 시스템 : Ultrasparc 5500
  - 운영체제 : Solaris2.7
  - SOAP 서버 : Tomcat 버전 4.0.1
- 지원 레지스트리
  - ebXML 레지스트리 : ebXML 버전 2.1 레지스트리 [13]
  - UDDI 레지스트리 : UDDI 버전 2.0 레지스트리 [16]
- 개발 언어
  - 구현 언어 : Java 버전 1.4.1
  - SOAP 파서 : Apache SOAP 2.2
  - XML 파서 : JAXP 버전 1.2_01

4.1 UDDI 서비스 미들웨어의 Publishing API 처리

4.1.1 Publishing API의 ebXML 서비스 변환

UDDI 서비스 미들웨어에서 Publishing API를 처리할 때 우선 UDDI API가 들어오면 이를 분석한 후 표 1에서 정의한 변환 알고리즘을 호출하게 되는데 이 알고리즘은 각 API에 따라 데이타 구조 매핑 정보를 참고하여 ebXML 서비스를 생성하게 된다.

그림 14는 UDDI Publishing API를 ebXML 서비스로 변환하는 알고리즘의 일부를 보인 것이다. 클라이언트가 보낸 Publishing 요구를 UDDI 서비스 미들웨어에서 받으면, UDDI 서비스 미들웨어는 convertPublishingAPI 알고리즘을 통해 API를 파싱해서 어떤 API인지 알아낸 후 각 API에 해당하는 ebXML 서비스 메시지를 생성한다. 이 때 UDDI API의 구조가 서로 다르고, ebXML 서비스로 변환하는 과정에서 UDDI API의 각 엘리먼트마다 ebXML의 어떤 엘리먼트로 매핑할지를 일일이 결정해야 하기 때문에 모든 UDDI API마다 서로 다른 코드를 구현해야 한다. 그림 14의

표 1 UDDI API 변환 알고리즘

알고리즘 이름	알고리즘의 기능
convertPublishingAPI	UDDI Publishing API를 ebXML 서비스로 변환
submitObjectsRequest	입력된 UDDI API에 대한 SubmitObjectsRequest 생성
updateObjectsRequest	입력된 UDDI API에 대한 UpdateObjectsRequest 생성
removeObjectsRequest	입력된 UDDI API에 대한 RemoveObjectsRequest 생성
authenticationToken	Publishing을 위한 토큰을 얻거나 없앰
convertInquiryAPI	UDDI Inquiry API를 AdhocQueryRequest로 변환
adhocQueryRequest	UDDI Inquiry API와 UDDI Publishing API중에 등록정보를 얻기 위한 API를 입력받아 ebXML 질의 서비스로 변환
convertResponse	UDDI API의 Response 변환 알고리즘

```
Algorithm : convertPublishingAPI (UDDI Publishing API를 ebXML 서비스로 변환)
Input : UDDI Publishing API
Output : ebXML 서비스
Steps :
Publishing API 파싱;
if API 타입 == save_XX || set_XX || add_XX then
   if 저장할 객체의 key가 이미 저장 then return UpdateObjectsRequest 생성(API);
   else return SubmitObjectsRequest 생성(API);
else if API 이름 == delete_XX then return RemoveObjectsRequest 생성(API);
else if API 이름 == XX_authToken then
   return authentication token 처리요구 생성(API);
else return AdhocQueryRequest 생성(API);
end if

Algorithm : submitObjectsRequest (save_service을 입력 받아 변환 수행)
Input : save_service
Output : SubmitObjectsRequest
Steps :
<SubmitObjectsRequest> 루트 엘리먼트 생성;
<Service>엘리먼트와 ebXML Publishing 템플릿 엘리먼트 생성후 자식으로 추가;
Organization 정보와 Association 생성;
while name이 존재
   <Name>엘리먼트를 생성후 자식으로 추가;
end while
while description이 존재
   <Description>엘리먼트 생성후 자식으로 추가;
end while
if bindingTemplate이 존재 then bindingTemplate 데이터구조 처리 알고리즘 실행;
if categoryBag이 존재 then categoryBag 데이터구조 처리 알고리즘 실행;
return SubmitObjectsRequest;
```

그림 14 UDDI Publishing API의 ebXML 서비스 변환 알고리즘

submitObjectsRequest 알고리즘은 이 중에서 save_service를 입력으로 받아서 ebXML의 SubmitObjectsRequest로 변환하는 부분이다. 이 알고리즘에서는 3.2절의 데이타구조 매핑에서 구성한 매핑 테이블을 참고하여 save_service의 모든 엘리먼트마다 이에 해당하는 SubmitObjectsRequest의 엘리먼트로 매핑하여 최종적인 ebXML 서비스를 생성한다.

4.1.2 응답 메시지의 변환

그림 15는 ebXML 레지스트리에서 응답한 응답 메시지를 UDDI API의 응답 메시지로 변환하는 알고리즘이다. UDDI 서비스 미들웨어는 ebXML 응답메시지를 받으면 파싱을 한 후에 성공적으로 서비스가 수행되었는지를 조사한다. 만약 Failure나 Unavailable인 경우 에러 메시지를 작성해서 UDDI 클라이언트에게 보내며, 성공하였을 경우 성공한 각 API에 대한 응답 메시지를 구성해서 보낸다. 또한 ebXML에서 Publishing에 대한 응답 메시지는 단순히 성공 여부만 알려주기 때문에 UDDI API에 대한 응답 메시지를 돌려줄 때는 UDDI API를 통해 저장할 때의 정보를 가지고 있다가 응답 메시지를 구성할 때 이 정보를 이용해야 한다.

```
Algorithm : convertResponse (UDDI API의 Response 변환)
Input : ebXML Response
Output : UDDI Response
Steps :
ebXML Response 파싱;
if response의 status == Success then
   각 API에 따른 응답 메시지 구성;
   return UDDI Response;
else if response의 status == Failure || Unavailable then
   return 에러메시지를 가지는 UDDI Response;
else if
```

그림 15 UDDI API에 대한 Response 변환 알고리즘

4.1.3 Publishing API 처리 예제

이 예제는 클라이언트가 Publishing API중에 한 가지인 save_service API를 UDDI 서비스 미들웨어에 보내고 처리하는 과정을 통해 본 연구에서 구현한 시스템이 실제로 어떻게 동작하는지를 기술한다.

(1) save_service API 호출

그림 16은 클라이언트에서 save_service API를 보내는 그림으로 이 클라이언트는 UDDI 서비스 미들웨어의 동작을 보이기 위해 만든 간단한 클라이언트로서 일반적인 UDDI클라이언트의 기능을 하지는 못한다.

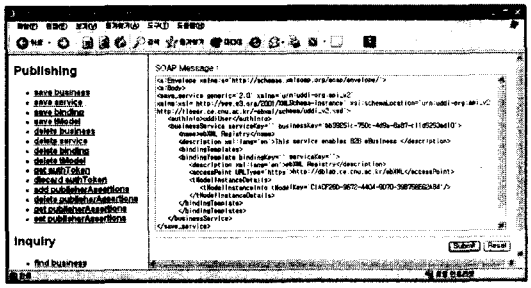

그림 16 save_service API의 예

(2) save_service API의 변환

그림 17은 실제 UDDI 서비스 미들웨어 안에서 save_service API 요구를 받아 그림 14의 submitObjectsRequest 알고리즘을 통해 ebXML의 SubmitObjectsRequest로 변환한 XML 예제이다. 그림과 같이 save_service의 service, name, description 엘리먼트는 각각 SubmitObjectsRequest의 service, name, description엘리먼트로 변환되었으며, bindingTemplate 엘리먼트는 ServiceBinding엘리먼트로 변환되었다.

```
<RootElement>
<SubmitObjectsRequest>
  <LeafRegistryObjectList>
    <Service id="servicekey" status="Approved" accessControlPolicy="uddiUser">
      <Name>
        <LocalizedString lang="en-us" value="ebXML Registry"/>
      </Name>
      <ServiceBinding id="" service="servicekey"
              accessURI="http://dblab.cs.cnu.ac.kr/ebXML" accessControlPolicy="uddiUser">
        <SpecificationLink specificationObject="externalLinkIdOverviewDoc"
              id="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4" accessControlPolicy="uddiUser"/>
      </ServiceBinding>
    </Service>
    <ObjectRef id="c38bf18d-e73e-4456-add5-ef4813482498"/>
    <Association id="BusServAssociationId0" associationType="BusinessEntity-BusinessService"
        sourceObject="c38bf18d-e73e-4456-add5-ef4813482498" targetObject="servicekey"/>
    <ExternalLink id="externalLinkIdOverviewDoc" externalURI="overviewURL">
      <Description>
        <LocalizedString lang="en-us" value="ExternalLinkDescription"/>
      </Description>
    </ExternalLink>
  </LeafRegistryObjectList>
</SubmitObjectsRequest>
</RootElement>
```

그림 17 그림 16의 save_service를 변환한 SubmitObjectRequest

(3) save_service API 응답

ebXML 레지스트리는 Pubilshing에 대한 응답으로 UDDI 서비스 미들웨어에게 그림 18과 같이 성공 여부만 알려주기 때문에 그림 19와 같은 UDDI 클라이언트가 인식하는 응답 메시지를 구성하기 위해서는 그림 15의 변환 알고리즘에서 Publishing API를 ebXML 서비스로 변환할 때 사용한 정보를 가지고 있다가 응답 메시지를 구성할 때 이용한다.

```
<RootElement>
  <RegistryResponse status ='Success'/>
</RootElement>
```

그림 18 SubmitObjectsRequest의 예

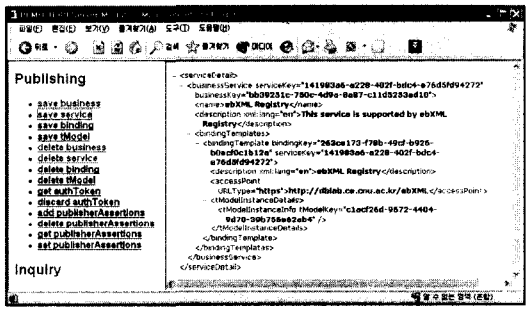

그림 19 save_service API의 응답 메시지인 serviceDetail의 예

4.2 UDDI 서비스 미들웨어의 Inquiry API 처리

4.2.1 Inquiry API의 ebXML 서비스 변환

UDDI 서비스 미들웨어에서 Inquiry API를 처리하기 위해서는 Publishing API와 마찬가지로 이를 분석한 후 표 1에서 정의한 변환 알고리즘을 호출하게 되며 이 알고리즘은 각 API에 따라 데이타 구조 매핑 정보를 참고하여 ebXML의 AdhocQueryRequest를 생성하게 된다.

그림 20은 UDDI Inquiry API를 처리하는 알고리즘의 일부분을 보인 것이다. UDDI 서비스 미들웨어는 Inquiry API를 ebXML 질의 서비스로 변환할 때 ebXML에서 필수적으로 지원해야 하는 FilterQuery로 변환하지만 UDDI의 serviceBinding, specificationLink에 해당하는 FilterQuery가 ebXML 서비스에 존재하지 않는다. 따라서 이 두 데이타구조를 질의하는 경우에는 ebXML에서 선택적으로 지원하는 SQL Query로 변환한다.

UDDI Inquiry API는 Publishing API와는 달리 하나의 UDDI API에서 주어지는 모든 검색조건을 처리할 수 있는 ebXML 질의 서비스가 없기 때문에 하나의 UDDI API를 하나의 ebXML 서비스로 변환할 수 없고 여러 개의 ebXML 질의로 변환한 후 각 질의의 실행

```
Algorithm : convertInquiryAPI (UDDI Inquiry API를 AdhocQueryRequest로 변환)
Input : UDDI Inquiry API
Output : UDDI Response
Steps :
Inquiry API 파싱;
Inquiry API에 대한 AdhocQueryRequest 생성(API);
실행결과 := AdhocQueryRequest 실행;
while 다른 데이터구조가 존재
      존재하는 데이터구조에 대한 AdhocQueryRequest 생성(API, 선행결과);
      AdhocQueryRequest 실행;
end while
return UDDI Response 생성;

Algorithm : adhocQueryRequest (find_service중 Service 검색을 위한 ServiceQuery
AdhocQueryRequest 생성)
Input : find_service
Output : AdhocQueryRequest
Steps :
<AdhocQueryRequest> 루트 엘리먼트 생성;
<ServiceQuery>엘리먼트와 ebXML 질의 템플릿 엘리먼트 생성후 자식으로 추가;
while name이 존재
   검색 조건 Clause를 포함한 <NameBranch>엘리먼트를 생성하고 자식으로 추가;
end while
if categoryBag이 존재 then categoryBag 데이터구조 처리 알고리즘 실행;
if tModelBag이 존재 then tModelBag 데이터구조 처리 알고리즘 실행;
return AdhocQueryRequest;
```

그림 20 UDDI Inquiry API 처리 알고리즘

결과를 통합해야 한다. 또한 여러 개의 ebXML 질의 서비스로 나누어서 질의를 할 경우 하나의 질의가 수행된 후에 나온 질의 결과를 가지고 다시 질의해야 하기 때문에 순차적으로 질의를 수행해야 한다. 따라서 Inquiry API를 처리하기 위해서는 convertInquiryAPI 알고리즘과 같이 우선 Inquiry 대상이 되는 데이타구조 검색 API를 ebXML 서비스로 변환해 질의를 수행한 후, 다른 데이타구조 정보가 필요한 경우 해당 ebXML 서비스를 추가로 생성하고 실행하게 된다.

예를 들어 find_service API는 businessService들의 리스트를 검색하는 API이지만 이 businessService와 연관된 businessEntity의 key값도 리턴해야 한다. 하지만 ebXML에서 businessService 정보를 검색하는 ServiceQuery에서는 businessEntity에 해당하는 Organization정보를 검색할 수 없기 때문에, 우선 ServiceQuery를 수행한 후에 검색된 서비스의 key와 연관된 Organization을 검색하기 위해서 OragnizationQuery를 수행해야 한다. 그림 20의 adhocQueryRequest 알고리즘은 UDDI API중에서 find_service를 ebXML의 AdhocQueryRequest로 변환하는 알고리즘이다. 이 알고리즘에서는 3.2절의 데이타구조 매핑에서 구성한 매핑 테이블을 참고하여 find_service의 모든 엘리먼트마다 이에 해당하는 AdhocQueryRequest의 엘리먼트로 매핑하여 최종적인 ebXML 질의 서비스를 생성한다. 또한 ebXML 질의 서비스의 구문이 UDDI API의 구문과 다르기 때문에 추가적으로 필요한 엘리먼트는 템플릿 형태로 가지고 있다가 매핑에 이용된다.

이외에도 Inquiry API는 질의 메시지에서 검색결과의 개수 제한이나 결과의 정렬 방식등을 설정하는 findQualifiers를 포함하고 있다. 하지만 ebXML 레지스트리에서 제공하는 FilterQuery는 검색결과의 개수제한이

나 결과의 정렬방식등에 대한 조건을 설정할 수 없다. 그러므로, UDDI 서비스 미들웨어는 findQualifiers를 만족시키는 결과의 작성을 위하여 결과메시지 작성기에서 결과에 대한 개수제한 및 정렬등을 수행한다.

이와 같이 본 연구에서 구현한 UDDI 서비스 미들웨어는 UDDI의 모든 API를 지원한다. 하지만 질의 API의 findQualifier중에서 대소문자 구별여부에 대한 조건인 caseSensitiveMatch는 처리하지 않는다. 현재 UDDI Inquiry API는 대소문자를 구별해서 질의를 수행할 수 있지만 ebXML에서는 대소문자를 구별하지 않아서 ebXML 레지스트리에서 처리할 수 없기 때문이다.

4.2.2 Inquiry API 처리 예제

이 예제는 클라이언트가 Inquiry API중에 한 가지인 find_service API를 UDDI 서비스 미들웨어에 질의하고 결과를 리턴하는 과정을 통해 본 연구의 시스템에서 질의가 실제로 어떻게 처리되는지를 기술한다. 또한 이전 절에서도 설명했듯이 find_service API의 경우 ServiceQuery와 OrganizationQuery 이렇게 두 개의 ebXML 질의를 수행해야 한다.

(1) find_service API 호출

그림 21은 UDDI 클라이언트에서 find_service API를 보내는 그림으로 publishing과 마찬가지로 텍스트 상자에 질의하고자 하는 API를 작성한 후에 서버에 이 API를 보내게 된다.

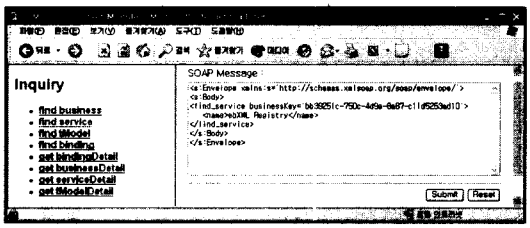

그림 21 find_service API의 예

(2) find_service API 변환 (ServiceQuery)

그림 22는 UDDI 서비스 미들웨어에서 그림 21의 find_service API를 받아 변환 알고리즘에 따라서 ServiceQuery로 변환한 예제이다. 이 FilterQuery는 4.2.1절에서 설명했듯이 find_service API 변환의 첫 번째 단계로서 Service를 검색하게 된다.

(3) ServiceQuery 질의 응답

ServiceQuery의 응답으로는 그림 23과 같이 질의 조건을 만족하는 Service들의 리스트가 리턴된다.

(4) find_service API 변환 (OrganizationQuery)

ebXML 레지스트리에서 Service정보가 리턴되면 이 Service정보와 관련된 Organization 정보를 검색하기

```
<RootElement>
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ServiceQuery>
      <NameBranch>
        <LocalizedStringFilter>
          <Clause>
            <SimpleClause leftArgument = "value">
              <StringClause stringPredicate = "Contains">ebXML Registry</StringClause>
            </SimpleClause>
          </Clause>
        </LocalizedStringFilter>
      <NameBranch>
    </ServiceQuery>
  </FilterQuery>
</AdhocQueryRequest>
</RootElement>
```

그림 22 businessSerive를 검색하는 ServiceQuery의 예

```
<RootElement>
<RegistryResponse status ='Success'>
  <AdhocQueryResponse>
    <FilterQueryResult>
      <ServiceQueryResult>
        <Service id="141983a6-a228-402f-bdc4-e76d5fd94272">
          <Name>
            <LocalizedString value="ebXML Registry"/>
          </Name>
        </Service>
        <Service id="0489bfb2-6404-44e8-8b86-ef63f242b67f">
          <Name>
            <LocalizedString value="ebXML Registry"/>
          </Name>
        </Service>
      </ServiceQueryResult>
    </FilterQueryResult>
  </AdhocQueryResponse>
</RegistryResponse>
</RootElement>
```

그림 23 ServiceQuery의 Response

위해 OrganizationQuery를 생성하는 변환 알고리즘을 통해서 다음과 같은 질의를 생성한다.

```
<RootElement>
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <OrganizationQuery>
      <SourceAssociationBranch>
        <ServiceQuery>
          <ServiceFilter>
            <Clause>
              <SimpleClause leftArgument="id">
                <StringClause stringPredicate="Equal">
                    141983a6-a228-402f-bdc4-e76d5fd94272</StringClause>
              </SimpleClause>
            </Clause>
          </ServiceFilter>
        </ServiceQuery>
      </SourceAssociationBranch>
    </OrganizationQuery>
  </FilterQuery>
</AdhocQueryRequest>
</RootElement>
```

그림 24 businessEntity를 검색하는 OrganizationQuery의 예

(5) OrganizationQuery 질의 응답

OrganizationQuery의 응답으로는 그림 25와 같이 질의 조건을 만족하는 Organization들의 리스트가 리턴된다.

(6) find_service API 응답

그림 26은 UDDI 서비스 미들웨어에서 find_service의 최종 결과를 UDDI response로 보낸 XML파일이다.

```
<RootElement>
<RegistryResponse status ='Success'>
  <AdhocQueryResponse>
    <FilterQueryResult>
      <OrganizationQueryResult>
        <Organization id="bb39251c-750c-4d9a-8a87-c11d5253ad10"
                primaryContact="179b28b0-d14a-4ccd-a833-b992c6da7222">
          <Address>Daejeon</Address>
          <TelephoneNumber>042-821-7721</TelephoneNumber>
        </Organization>
      </OrganizationQueryResult>
    </FilterQueryResult>
  </AdhocQueryResponse>
</RegistryResponse>
</RootElement>
```

그림 25 OrganizationQuery의 Response

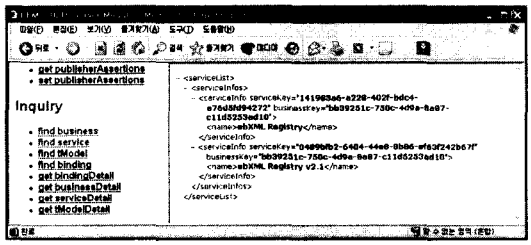

그림 26 find_service response의 예

Publishing에 대한 Response를 구성할 때는 ebXML 레지스트리에서 성공여부만 알려주기 때문에 클라이언트가 요구한 Publishing API 정보를 이용하지만 Inquiry에서는 ebXML 응답메시지에 검색결과가 리턴되기 때문에 이 정보를 가지고 UDDI Response를 구성한다.

5. 관련 연구

5.1 성능 분석 및 비교

이번 절에서는 ebXML 레지스트리에서 ebXML 서비스뿐만 아니라 UDDI 서비스까지 처리할 수 있도록 하는 UDDI 서비스 미들웨어의 성능을 비교하기 위해서 다음과 같은 비교를 수행한다. 즉, ebXML 레지스트리가 존재하는 경우에 추가적으로 UDDI 서비스를 지원하기 위해서 본 연구의 미들웨어를 도입했을 경우와 UDDI 레지스트리를 추가적으로 도입했을 경우를 분석한다.

표 2와 같이 UDDI 서비스를 지원하기 위해서 UDDI 서비스 미들웨어를 도입하면 UDDI 레지스트리를 새로 구축하는 것보다 비용과 노력 절감에 효과가 있음을 알 수 있다. 이러한 장점은 UDDI 레지스트리를 구축하려면 하나의 서버를 구축하는 것과 같기 때문에 이에 따르는 하드웨어 장비, UDDI 레지스트리와 데이타베이스 라이센스 비용 그리고 관리 인원과 정책들이 필요하지만, 본 연구에서 개발한 미들웨어는 UDDI API를 ebXML 서비스로 변환하는 라이브러리로 제공될 수 있기 때문에 기존의 ebXML 레지스트리 서버에 하나의 모듈로 추가됨으로 인해 새로운 시스템을 구축하는 것보다 비용과 노력이 절감되며 데이타베이스 접근도 ebXML 레지스트리 서버에서 담당하기 때문에 이에 대한 고려를 하지 않아도 된다. 따라서 현재 대표적인 전자상거래 레지스트리로 사용되는 ebXML 레지스트리와 UDDI 레지스트리를 모두 도입하기 위해서 각각의 레지스트리를 따로 도입하기보다는 UDDI 레지스트리보다는 일반적인 구조와 기능을 가지는 ebXML 레지스트리를 도입하고 본 연구에서 개발한 UDDI 서비스 미들웨어를 추가함으로서 전자상거래 하려는 기업들이 적은 비용과 노력으로 두 개의 솔루션을 모두 가질 수 있는 이점을 가지게 된다.

5.2 JAXR(Java API for XML Registries)

현재 본 연구에서 제안한 문제와 이에 대한 해결방법들을 수행한 연구는 존재하지 않지만 비슷한 연구로서 JAXR(Java API for XML Registries)이 있다. JAXR은 SUN에서 UDDI와 ebXML을 통합하기 위해서 제공하는 데이타 모델로써 UDDI 레지스트리와 JAXR의 데이타에 대한 매핑과, ebXML 레지스트리와 JAXR과의 데이타 매핑을 각각 정의하고 있으며, 또한 이 두 개의 레지스트리에 클라이언트가 모두 접근할 수 있도록 JAXR 자바 API를 제공한다.

하지만 JAXR과 본 연구에서 구현한 UDDI 서비스 미들웨어는 다음과 같은 차이점 있다. 본 연구에서는

표 2 UDDI 서비스 미들웨어와 UDDI 레지스트리 구축 비교

	UDDI 서비스 미들웨어 구축	UDDI 레지스트리 구축
ebXML 서비스 지원	지원	지원
UDDI API 추가지원	지원	지원
구현 시스템	UDDI 서비스 미들웨어 라이브러리만 추가	UDDI 레지스트리 시스템 추가 구축
데이타베이스 관리	필요 없음	UDDI 레지스트리에 필요
라이센스	라이브러리 라이센스만 필요	UDDI 레지스트리와 데이타베이스 라이센스가 추가적으로 필요
하드웨어 장비	필요 없음	UDDI 레지스트리에 필요
관리 인원	필요 없음	UDDI 레지스트리에 필요
시스템 관리 정책	필요 없음	UDDI 레지스트리에 필요

UDDI 클라이언트가 ebXML 레지스트리 접근을 위한 추가적인 모듈 없이 마치 ebXML 레지스트리를 UDDI 레지스트리처럼 사용하게 하지만, JAXR에서는 UDDI 클라이언트가 UDDI 레지스트리뿐만 아니라 ebXML 레지스트리에 접근하기 위해서 JAXR API를 이용해야 한다. 또한 JAXR은 두 레지스트리의 공통되는 구조만 매핑시킴으로써 본 연구에서 제시한 불일치 문제들이 발생하지는 않지만 공통되는 부분만 매핑하고 서로 일치하지 않는 부분은 사용하지 않기 때문에 UDDI 또는 ebXML 레지스트리의 모든 기능을 수행할 수 없다. 이와 같이 JAXR은 ebXML과 UDDI 레지스트리들을 단일하게 접근하기 위한 방법을 제공한다는 점에서 본 연구와 비슷하지만 접근 방법이 서로 다르기 때문에 직접적인 비교는 힘들다.

만약 차후에 본 연구와 비슷한 시스템이 개발된다면 본 논문에서 개발된 시스템과의 비교가 필요할 것이다. 이를 위해서는 본 시스템과 새로 개발되는 시스템의 알고리즘 비교를 통해 이론적인 성능을 분석하고, 이를 바탕으로 동일한 환경에서 실제 시스템의 성능을 비교함으로써 보다 나은 연구가 수행될 수 있을 것이다.

6. 결 론

본 논문에서는 UDDI 레지스트리 클라이언트가 ebXML 레지스트리에 대하여 UDDI 레지스트리 서비스를 요구할 수 있도록 하는 UDDI 서비스 미들웨어를 설계하고 구현하였다. 이를 위해 ebXML RIM과 UDDI의 데이타 구조를 분석해서 ebXML RIM과 UDDI의 데이타 구조를 매핑하였으며, 이들 매핑정보를 바탕으로 UDDI API를 ebXML 서비스로 변환하여 처리하는 UDDI 서비스 미들웨어를 동작 메커니즘 개발하였다. 또한 ebXML RIM과 UDDI의 데이타 구조를 매핑할 때 각 모델의 구조적인 차이로 인해 필수 제약조건 불일치와 데이타 크기 제약조건 불일치가 발생하기 때문에 본 논문에서는 이러한 불일치를 분류하고 이에 대한 해결방법을 제시하였다.

본 연구에서는 위와 같은 UDDI 서비스 미들웨어를 개발함으로써 다음과 같은 장점을 제공한다.

- UDDI 서비스 미들웨어는 ebXML 레지스트리에 별도의 변경을 가하지 않고도 UDDI 레지스트리의 API를 처리할 수 있다.
- ebXML 레지스트리를 이용하여 ebXML 레지스트리 서비스뿐만 아니라 UDDI API 서비스를 제공할 수 있다.
- ebXML 도입비용으로 웹 서비스 또한 추가적으로 도입할 수 있으므로, 비용 및 노력의 절감효과를 제공한다.

본 연구에서 구현한 UDDI 서비스 미들웨어는 모든 UDDI 서비스를 지원한다. 하지만 UDDI 레지스트리와 ebXML 레지스트리에서 대소문자를 구별 방법이 다르기 때문에 현재 이에 대한 지원을 하지 못하고 있으며 앞으로 이 문제를 처리하기 위한 연구가 필요하다.

본 시스템은 ebXML 레지스트리 버전 2.1과 UDDI 버전 2.0을 지원하지만 현재 UDDI와 ebXML 레지스트리 버전 3.0의 표준화 작업이 진행중이다. 또한 UDDI 레지스트리 버전 3.0과 ebXML 레지스트리 버전 3.0은 backward compatibility를 지원하기 때문에 본 연구에서 수행한 결과를 바탕으로 UDDI와 ebXML 버전 3.0의 데이타구조 분석하고 UDDI 레지스트리에서 추가된 API에 대한 변환 메커니즘 설계,구현하면 UDDI 버전 3.0에 대한 지원이 가능할 것이다.

참 고 문 헌

[1] UN/CEFACT, OASIS, *"ebXML (electronic business eXtensible Markup Language),"* http://www.ebxml.org

[2] Microsoft, IBM, Ariba, *"UDDI (Universal Description, Discovery and Integration) Version 2.0,"* http://www.uddi.org/specification.html

[3] UN/CEFACT, OSAIS, *"Registry Information Model v2.1,"* http://www.oasis-open.org/committees/regrep/documents/2.1/specs/ebrim_v2.1.pdf

[4] UN/CEFACT, OSAIS, *"Registry Services Specification v2.1,"* http://www.oasis-open.org/committees/regrep/docu ments/2.1/specs/ebrs.pdf

[5] UN/CEFACT, OSAIS, *"Message Service Specification v2.0,"* http://www.ebxml.org/specs/ebMS2.pdf

[6] W3C notes, *"Simple Object Access Protocol (SOAP) 1.1,"* http://www.w3.org/TR/SOAP

[7] W3C notes, *"Web Services Description Language (WSDL)1.1,"* http://www.w3.org/TR/SOAP

[8] UN/CEFACE, OSAIS, *"Using UDDI to Find ebXML Registry/Repository,"* ebXML white paper, Apr. 2001.

[9] Shalom Tsur, Serge Abiteboul, Rakesh Agrawal, Wmeshwar Dayal, Johannes Klein, Gerhard Weikum, *"Are Web Services the Next Revolution in e-Commerce?,"* In Proc. of Very Large Data base(VLDB), pp.614-617, Sep. 2001.

[10] 김미혜, 이경하, 이규철, "웹 서비스 등록/검색 도구 기능 정의", 29th 한국정보과학회 가을 학술발표논문집, pp.106-108, Oct. 2002.

[11] 전희영, 이규철, "ebXML 등록기/저장소 객체 질의 관리 시스템의 설계 및 구현", 한국전자거래학회지, Vol.7, No.3, pp.21-48, Dec. 2002.

[12] 조강녕, 이병현, 김상균, 이규철, "ebXML Registry v2.0 서버 개발", e-Korea를 위한 전자거래 종합학술대회, pp.356-363, Sep. 2002.

[13] 조강녕, "ebXML 레지스트리 v2.1 서버 개발", 석사학위논문, 충남대학교, 2003.
[14] 박재홍, "ebXML 레지스트리 기반의 UDDI 레지스트리 서비스 미들웨어 설계 및 구현", 석사학위논문, 충남대학교, 2003.
[15] 박재홍, 김상균, 이규철, 조현규, "ebXML Registry 기반의 UDDI Registry Server 구현", 18th 한국정보처리학회(KIPS) 추계 학술발표논문집, pp.1749-1752, Nov. 2002.
[16] 김영선, 유수진, 이경하, 이규철, "UDDI 2.0 레지스트리 설계 및 구현", 29th 한국정보과학회 가을 학술발표논문집, pp.111-111, Oct. 2002.

박 재 홍

2000년 충남대학교 공과대학 컴퓨터공학과(공학사). 2003년 충남대학교 공과대학 컴퓨터공학과(공학석사). 2001년 MP3GASA.COM 연구원. 2000년~현재 KT 운용시스템연구소 프리랜서. 관심분야는 데이타베이스, XML, LBS, CBD, 전자 상거래

김 상 균

1999년 충남대학교 공과대학 정보통신공학과(공학사). 2001년 충남대학교 공과대학 컴퓨터공학과(공학석사). 2001년~현재 충남대학교 공과대학 컴퓨터공학과 박사과정. 2003년~현재 (주)K4M 연구원. 관심분야는 데이타베이스, XML, e-비즈니스

이 규 철

1984년 서울대학교 공과대학 컴퓨터공학과(공학사). 1986년 서울대학교 공과대학 컴퓨터공학과(공학석사). 1990년 서울대학교 공과대학 컴퓨터공학과(공학박사) 1994년 미국 IBM Almaden Research Center 초빙 연구원. 1995년~1996년 미국 Syracuse University, CASE Center 초빙 교수. 2000년~현재 한국 ebXML 전문위원회 위원장. 2001년~현재 전자상거래 표준화 통합 포럼 전자거래 기반 기술위원회 위원장. 2003년~현재 한국전자거래학회 편집이사. 2003년~현재 웹 코리아 포럼 웹 기술분과 위원장. 현재 충남대학교 공과대학 컴퓨터공학과 교수. 관심분야는 데이타베이스, XML, 정보 통합, e-비즈니스

김 경 일

1999년 충남대학교 컴퓨터공학과(공학사) 2001년 충남대학교 컴퓨터공학과(공학석사). 2001년~현재 한국전자통신연구원 지능형서비스플랫폼연구팀 연구원. 관심분야는 전자상거래, 데이타베이스, 시맨틱 웹

김 록 원

1998년 충북대학교 컴퓨터공학과(공학사) 2000년 충북대학교 전산학과(이학석사) 2000년~현재 한국전자통신연구원 지능형서비스플랫폼연구팀 연구원. 관심분야는 전자상거래, 데이타베이스, 시맨틱 웹

송 병 열

1995년 전북대학교 전자공학과 학사. 1997년 전북대학교 전자공학과 석사. 1997년~현재 한국전자통신연구원 지능형서비스플랫폼연구팀 연구원. 관심분야는 전자상거래, 지식관리시스템, 인공지능

조 현 규

1988년 한국외국어대학교 독일어문학과(문학사). 1990년 고려대학교 대학원 경영학과 경영정보전공 석사. 1997년 한남대학교 대학원 경영학과 경영정보 전공 박사. 1988년~1990년 현대해상 정보시스템부. 1990년~현재 한국전자통신연구원 지능형서비스플랫폼연구팀 팀장. 관심분야는 전자상거래, 경영정보시스템, 지식관리시스템