工學碩士學位論文

이기종 분산 환경에서 웹 서비스를 위한 도메인 객체 인터페이스 기반의 프레임워크

A Framework based on Domain Object Interface for Web Services in heterogeneous Distributed Environments

順天大學校大學院

멀티미디어工學科

林 銀 天

2008年 12月

이기종 분산 환경에서 웹 서비스를 위한 도메인 객체 인터페이스 기반의 프레임워크

指導敎授 沈 春 甫

이 논문을 工學碩士學位 論文으로 提出함

2008 年 12 月 21 日

順天大學校大學院

멀티미디어工學科

林銀天

林銀天의 工學 碩士 學位
論文을 認准함

審査委員長　　吳　源　根　　(인)

委　　員　　李　成　根　　(인)

委　　員　　沈　春　甫　　(인)

2008 年　12 月　21 日

順天大學校大學院

目　　次

그림 목차

표 목차

요약

이기종 분산 환경에서 웹 서비스를 위한 도메인 객체 인터페이스 기반의 프레임워크

임 은 천
멀티미디어 공학과
순천대학교
지도교수. 심 춘 보

객체 지향 소프트웨어는 다수의 객체를 선언하고 객체끼리의 통신을 하며 동작한다. 일반적으로 분산 환경에서의 객체 지향 소프트웨어는 웹 서비스를 통해 구축된다. 분산 환경에서의 성공적인 웹 서비스 구현을 위해서는 서비스의 검색 가능성, 유효성, 신뢰성, 안정성, 가용성, 도메인 객체로의 접근성, 실행 성능 등이 요구된다.

현실적으로 웹 서비스 구현을 위해서 모든 기반 기능을 서비스 도메인 개발자가 구현하기 어렵기 때문에 프레임워크를 이용하여 웹 서비스를 구현하는 것에 대한 연구가 계속 되고 있다. 기존의 웹 서비스 프레임워크에 대한 연구들은 웹 서비스의 등록 및 검색, 통신 프로토콜, 조합, 실행 분야 등에서 발생되는 문제점에 대한 전반적인 해결책을 제시하고 있다. 그러나 웹 서비스를 통해서 비즈니스 로직을 다룰 경우에는 표준과 관련된 많은 것을 고려해야 한다. 여기서 문제는 웹 서비스를 구현할 때 사용하는 프레임워크와 기존의 데이터 소스를 연결하고 이 데이터 소스를 레거시 웹 응용 프로그램에 보여줄 때 발생한다. 즉, 도메인 객체와 데이터 소스의 매핑, 로컬 비즈니스 로직 간의 연결, 로컬 비즈니스 로직의 웹 서비스 설정 및 배포, 실행의 복잡성 문제가 대두된다. 기존 개발 모델을 그대로 사용할 수 없기 때문에 추가적인 복잡도가 요구되며 이는 기존 개발 모델의 변동을 최소화하는 경량의 프레임워크를 요구하게 된다.

본 논문에서는 웹 서비스에 대한 검색, 설정, 실행 편의성을 도모한 웹 서비스 메타 모델을 제시하고 이를 처리하기 위한 도메인 객체 인터페이스(Domain Object

Interface, 이하 DOI)를 제안하며, 웹 서비스 설정, 배포 및 실행을 위한 경량의 DOI 기반의 프레임워크(Distribution Oriented Knowledge-base of Dynamically Operable Web Services, 이하 DOKDO-WS)를 설계 및 구현한다. 제안하는 DOKDO-WS 프레임워크는 도메인에서 사용되는 비즈니스 로직을 등록 및 검색하기 위한 웹 서비스 레지스트리를 제공한다. 웹 서비스 레지스트리에 서비스를 확인 및 실행할 수 있는 웹 서비스 브라우저 기능을 제공하며, 서비스 실행 계층과 뷰 계층에서 서비스를 동적 혹은 정적으로 조합하는 기능을 제공한다. 서비스는 검증을 통과한 경우에만 추가되므로 런타임에는 실행 가능한 서비스만 보이게 된다. 레지스트리 상의 서비스들은 리플렉션을 통해서 로컬 서비스를 실행되고, SOAP 프록시 인스턴스를 통해서 원격 서비스를 실행되며 웹 브라우저를 통해서 실행된다. 아울러 DOKDO-WS 프레임워크의 메타 모델은 XML 스키마로 선언되며 데이터 메타 모델과 서비스 메타 모델로 나뉜다. 서비스 메타 모델은 로컬, 원격 서비스를 선언한다. 데이터 메타 모델은 테이블 및 뷰를 생성하고 대부분의 퍼시스턴스 연산에 대해서 높은 성능을 보이는 저장 프로시저를 자동으로 생성하기 위해서 선언된다. XML 변환기를 통해서 도메인 객체의 데이터가 서비스 사용자에게 전달되며 서비스 사용자에게 XSLT 기반의 처리를 통해서 적합한 형태의 GUI를 보이게 된다.

프레임워크가 가진 효율성, 안정성, 접근성 측면을 확인하기 위해서 서비스 레지스트리의 성능을 연산 시간을 기준으로 평가하고 DOI의 처리 능력을 평가하기 위해서 은행 응용에서의 트랜잭션 처리에서의 연산에 대한 성능 평가를 수행한다. 서비스 레지스트리에서 로컬 서비스 등록, 원격 서비스 등록, 서비스 수정, 서비스 삭제의 평균 연산 시간은 평균적으로 각각 59.0646ms, 40.2767ms, 114.2569ms, 52.14263ms가 소요되며, 단일 서비스 검색 연산은 해싱 기반의 검색으로 평균 0.0012ms가 소요된다. 전체 서비스 나열 연산은 서비스의 수에 비례해서 미세하게 증가하거나 감소하며 10개의 서비스를 등록한 상태에서 전체 서비스 나열은 평균 0.03212ms의 시간이 소요된다. 도메인 객체 처리 성능 평가에서는 DBMS, ORM 프레임워크, 퍼시스턴스 연산의 종류를 기준으로 테스트를 수행하며 MS-SQL, Oracle DBMS에서 삽입, 수정, 삭제, 복합, 선택 연산에서 제안한 DOI가 가장 우수한 성능을 보인다. 특히 웹 서비스 환경에서 자주 사용되는 선택 연산의 경우 다른 프레임워크에 비해 높은 성능을 보인다. 그러나 MySQL DBMS에서는 복합 연산, 선택 연산에서 매우 낮은 성능을 보이는데 이것은 테스트한 JDBC 드라이버의 결점에 의한 것으로 C 언어에서 평가를

할 경우 다른 ORM 프레임워크 보다 5배 정도 빠른 것을 알 수 있다. 10 만 번의 퍼시스턴스 연산 실행 과정에서 에러는 평균 0~3개 정도 발생한다. 성능 평가를 바탕으로 서비스 레지스트리, 실행 엔진으로써 DOKDO-WS가 높은 효율성, 안정성, 신뢰성을 가지는 것을 검증한다.

키워드 : 웹 서비스 프레임워크, 웹 서비스 레지스트리, 웹 서비스 조합, 웹 서비스 실행 엔진, 도메인 객체, ORM, SOA

I. 서 론

1. 개요

비즈니스 로직의 결합을 감소시키며 분산 환경에서 각 서비스 간 상호 작용을 위해서 여러 기술이 발전과 쇠퇴를 거듭하고 있다. 과거의 분산 환경 기술은 사용자에게 위치의 투명성, 상호 호환성을 위한 독자적인 모델을 발전시켜왔다. 그러나 특정 개발 환경에 종속적이지 않은 환경을 만들기 어렵고 분산 객체 이용자가 플랫폼 별로 검색을 위한 별도의 인터페이스를 정의하고 브로커를 통해 객체에 대한 메시지를 전달해야 했다. 게다가 세션 관리, 보안 문제와 같은 도메인에 밀접한 서비스는 분산 객체 이용자 자신만의 방식으로 설계하고 구현해야 했다. 이런 문제를 해결하기 위해서 분산 환경 구현을 위한 XML 기반의 표준 웹 서비스가 등장한다.

웹 서비스는 과거 분산 환경 기술을 구현하는데 쓰인 개발 방법론인 CBD (Component Base Development) 방식을 서비스의 관점에서 확장한 SOA(Service Oriented Architecture) 기반으로 설계되고 구현된다. 웹 서비스 분야는 등록 및 검색[1-10], 프로토콜[5, 9, 11-13], 조합[14-26], 실행[23-30]이라는 주요 연구 분야를 파생시켰다. 그리고 이런 세부 분야들을 통합하여 서비스 제공자에게 쉬운 개발 환경을 제공하는 프레임워크[31-43]에 대한 연구가 계속 되고 있다.

하나의 웹 서비스 응용은 도메인 모델을 가지고 모델링을 수행한다. 도메인 객체는 각각의 도메인 모델을 실제 비즈니스 로직에 적용하는데 이용된다. 도메인 객체는 실행 시점에 사용되기 위해서 저장소를 사용하기 때문에 저장소에 대한 모델링을 수행할 필요가 있다. 또한 실행 시점의 도메인 객체는 사용자에게 정보 전달을 위해서 또 다른 형태로 변형되어야 한다. 그러므로 웹 서비스 프레임워크는 도메인 객체의 표현, 전달, 수정에 대한 연구를 추가적으로 할 필요가 있다. 도메인 모델이 시멘틱 온톨로지로 변화되더라도 서비스 제공자는 도메인 객체 모델링에 대한 문제에서 벗어날 수 없다[44-49]. 표준 웹 서비스 기술에서 도메인 모델을 XML로 표현하기 때문에 XML과 객체와의 매핑에 대한 연구도 계속되고 있다[49].

제안하는 DOKDO-WS 프레임워크는 동적 웹 서비스 추가, 수정, 삭제를 제공하는 웹 서비스 레지스트리를 제공한다. 웹 서비스는 웹 서비스 브라우저를 통해서 확인 및 수정이 가능하며 제안한 메타 모델을 기준으로 저장된다. 성능을 위해서 메타 모델을 담은 도메인 객체는 주기억 장치에 상주하며 서비스 실패로 인한 데이터 손실을 최소화하기 위해서 주기적으로 퍼시스턴스 영역에 스스로를 저장한다. 메타 모델 도메인 객체에 저장된 서비스는 주기억 장치에 상주되는 시점에 검증되기 때문에 실행 시간에는 실행 가능한 서비스만 보이게 된다.

DOKDO-WS는 동적 서비스 조합을 지원한다. 먼저 서비스 뷰 계층에서의 조합은 개별 서비스를 묶은 함수 그룹을 조합한다. 이와 다르게 서비스 실행 계층에서의 조합은 컴포넌트 조합, AOP, 동적 프록시 생성의 방법으로 조합한다. 단일의 혹은 조합된 연산은 묶여서 단위 서비스를 만든다. 외부적으로는 웹 서비스는 로컬 원격 구분 없이 웹 브라우저, SOAP 기반의 프록시를 통해서 실행될 수 있다. 내부적으로는 로컬 서비스는 서비스 인스턴스로 직접 실행 메시지를 전달하여 실행하고, 원격 서비스는 프레임워크가 SOAP 기반의 프록시를 생성해서 실행한다. 프레임워크는 부분적으로 세션을 지원하며 기존의 개발 환경을 변경하지 않고 웹 서비스를 구현할 수 있는 방법론을 제공한다.

비즈니스 로직에서 사용되는 도메인 객체의 데이터 모델은 ER 기반의 데이터 모델이다. 선언된 데이터 모델은 자동적으로 RDBMS에 테이블 및 뷰를 생성하고 이를 다루기 위한 저장 프로시저를 생성한다. 비즈니스 로직에서의 퍼시스턴스 연산은 자동 생성된 저장 프로시저를 통해서 이뤄지거나 서비스 제공자가 직접 선언한 저장 프로시저를 통해서 이뤄진다. 서비스 제공자는 최소한의 선언과 API 이용을 통해서 비즈니스 로직에서 퍼시스턴스 영역에 쉽고 빠르게 접근할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 웹 서비스의 등록 및 검색, 조합, 실행, 프로토콜, 프레임워크와 관련된 기존 연구들과 도메인 객체의 정의, 퍼시스턴스 계층과의 연결에 관련된 기존 연구들을 살펴본다. 3장에서는 이기종 분산 환경에서 웹 서비스를 위한 DOKDO-WS 프레임워크를 기술한다. 4장과 5장에서는 구현 및 성능 평가를 기술하고 결론 및 향후 연구 방향을 제시한다.

2. 연구 동기

웹 서비스 제공자는 이용이 편리하고 확장성이 높은 프레임워크를 필요로 한다. 이용의 편리성과 확장성은 서비스 제공자에게 기존의 서비스 개발 환경의 인프라 구조를 재사용하는 개발 방법론을 제공하고, 표준 웹 서비스 기술과 호환성을 제공함으로써 획득할 수 있다.

프레임워크는 서비스를 확인하고 동적으로 실행 및 조합할 수 있어야 한다. 프레임워크가 제공하는 서비스 레지스트리에서 별다른 제약 없이 원격과 로컬의 웹 서비스를 등록, 검색, 변경할 수 있어야 한다. 일반적으로 서비스 제공자는 특정 도메인 모델에 정확하게 일치하는 서비스만을 제공하길 원하기 때문에 실행 시점에 동적으로 서비스를 확인하고 변경할 수 있는 기능은 필수적이다.

프레임워크는 프로토콜을 쉽게 다루기 위한 XML을 위한 인터페이스를 제공해야 한다. 웹 서비스 외부에서 전달되는 메시지는 HTTP 기반의 SOAP을 이용하여 전달된다. SOAP 내부의 메시지는 XML 기반의 프로토콜을 이용한다. 서비스 제공자는 XML을 위한 인터페이스를 이용하여 SOAP 내부에 원하는 프로토콜을 선언하고 메시지를 전달할 수 있다.

실행에서 상태의 유지는 중요한 문제다. 특히 기존의 웹 브라우저를 이용해서 웹 서비스를 실행할 때 상태를 유지하기 위해서는 세션에 대한 개념을 필요로 한다. 입력, 수정, 삭제와 같은 단순 연산은 무상태 서비스 인스턴스로 처리해도 문제가 없다. 그러나 서비스에서 다뤄지는 자원에 대한 접근 권한을 확인하거나 이전 접근과의 연관성을 획득하는 문제를 해결하기 위해서는 상태가 있는 연산 객체를 필요로 한다.

마지막으로 프레임워크는 도메인 모델링을 쉽게 할 수 있는 방법을 제공해야 한다. 어떠한 응용이든지 언제나 도메인 모델링을 필요로 한다. 비록 웹 서비스에서 이용할 수 있는 도메인 모델이 분산되어 있더라도 비즈니스 로직 계층에서는 단일 서비스 도메인에 묶여서 처리한다. 이 과정에서 도메인 객체, 저장소, 표현 언어로의 매핑 문제는 자동화될 필요가 있다.

대부분의 웹 응용은 모델을 저장할 때 DBMS를 사용하기 때문에 SQL을 작성하고 수정하는 작업은 반복적이다. 물론 도메인에 특징적인 프로세스는 자동화할 수 없고 직접 개발할 수밖에 없다. 입력, 검색, 수정, 삭제와 같은 반복적인 퍼시스턴스 연산은 자동적으로 생성될 수 있고, 모델과 표현 언어로의 매핑도 자동화 할 수 있다. 이는 결과적으로 비즈니스 로직과 도메인 모델의 결합도를 크게 낮추게 된다.

II. 관련 연구

제안된 프레임워크는 기존의 웹 서비스에 대한 연구와 도메인 객체에 대한 폭넓은 이해를 필요로 한다. 그렇기 때문에 각 연구들에서 언급했던 문제점들을 이 장을 통해서 세밀하게 살펴보고 이를 기반으로 주요 문제점들을 도출한다.

1. 웹 서비스

1) 웹 서비스 등록 및 검색

웹 서비스 이전의 분산 객체 기술에서는 중앙 집중 방식인 브로커링 및 브리징을 통해서 서비스를 검색하고 실행했다. 그러나 웹 서비스 기반의 환경에서는 웹 서비스 저장소 또한 분산 시키고 Match-Making 기반의 검색 알고리즘을 사용해야 한다[1]. 표준 웹 서비스 저장소를 구현하는 과정은 UDDI를 구현하는 과정이다[3]. 웹 서비스의 분산성은 위치의 투명성 또한 요구하지만 서비스 실행 엔진의 성능 향상을 위해서 웹 서비스들의 로컬과 원격의 구분이 필요하다. 로컬 서비스는 원격 서비스보다 빠른 성능과 높은 신뢰성을 보장하기 때문에 도메인에서 중요한 서비스는 가능하다면 로컬 서비스로 옮기는 것이 좋다. 로컬 서비스는 컴포넌트 레벨에서 접근이 가능하기 때문에 UDDI 기반의 등록 및 검색을 이용할 필요는 없다. 이는 오히려 성능 하락을 발생시킬 수 있다. 이외에도 표준으로 제시된 UDDI의 단점은 매우 추상적인 표준, 기본적인 검색 기능, 등록된 서비스의 품질에 대한 보장을 하지 않음, 서비스를 캐싱하지 않음 등이다[6].

서비스는 주로 OWL-S 컴포넌트 중 서비스 프로파일 컴포넌트에 존재하는 IOPE(Input, Output, Precondition, Effect)에 의해서 검색이 된다. [10]에서는 정확히 일치하는 것을 검색하는 알고리즘 및 자료구조가 제시되었다. 도메인 변경에 따라서 동일한 단어라도 다르게 해석되기 때문에 도메인 변경에 따른 추론을 위해서 시멘틱 웹 개념에 적합한 알고리즘 및 자료구조가 제시되었다[7, 8, 15]. 웹 서비스 사용자가 반드시 IOPE만을 고려하는 것이 아니라 부가적인 메타 정보를 기준으로 서비스를 검색을 할 수도 있다. 이런 정보는 서비스 등록, 실행, 변화 요청의 시점에 SOAP 메시지

추가되거나 수정될 수 있다[5, 9]. 이런 정보를 기준으로 프레임워크는 검색을 수행할 수 있다. 그러나 웹 서비스 프레임워크 내에서 전문적인 검색 기능을 구현하는 것보다 전문적인 에이전트, 웹 크라울러 혹은 검색 엔진을 통한 검색 방법이 보다 효율적이며 현실적이다[4, 6]. 그러나 서비스 검색이 완료되었고 서비스가 실행되었더라도 검색된 서비스의 품질에 대한 어떠한 보장도 할 수 없다. 그 중 가장 주요한 원인은 서비스 레지스트리가 정적일 경우 시간이 갈수록 실행이 안 되는 원격 서비스가 많아질 가능성이 높다는 것이다. 원격 서비스는 주 기억장치에 상주 시킬 때 검증을 해야 하며 로컬 서비스의 비즈니스 로직이 변경되면 이를 자연적으로 적용할 수 있어야 한다[17]. 서비스 레지스트리에 요구되는 특징은 다음과 같이 정리된다.

첫째, 서비스에 대한 메타 데이터에는 검색을 위한 서비스에 대한 설명이 반드시 필요하다. 서비스 제공자는 서비스를 통해서 얻는 이윤이 낮다면 자세한 설명을 적지 않을 수도 있다. 반면 서비스를 통해서 얻을 수 있는 이윤이 많다면 더 자세한 설명을 검색 엔진에 노출하기 원할 것이다.

둘째, 서비스 사용자는 비 기능적 요소에 대한 자동화된 검사를 필요로 할 수 있다. 대다수의 서비스 사용자는 높은 성능을 가지는 서비스에 더 높은 우선순위를 주기 때문에 기본적으로 서비스에 대한 성능을 알려줄 수 있는 메타 데이터가 필요하다.

셋째, 같은 도메인 내의 웹 서비스가 서비스 컨테이너 혹은 서버와 생존 주기가 같아야 한다. 컨테이너의 초기화 시점에 서비스에 대한 메타 데이터를 도메인 객체에서 이용할 수 있도록 초기화하고, 컨테이너나 서버가 종료되면 외부에서 검색 및 사용이 불가능해진다. 이는 실행 시간에 언제나 서비스를 검색하고 실행 가능한 상태로 유지하는 것을 보장해준다.

넷째, 로컬 서비스에 대한 종점은 자동으로 생성되어야 한다. 이는 잘못된 실행이 원천적으로 발생하지 않도록 해주며, 서비스 메타 데이터 생성과 수정에서 발생하는 오버헤드를 없애준다. 즉 2차적인 저장소를 필요로 하지도 않고 메타 데이터에 대한 모델링을 필요로 하지 않게 된다.

다섯째, 현재 도메인 내에서 실행 가능한 서비스를 조회할 수 있는 서비스 맵에 대해 접근할 수 있는 공통된 인터페이스를 제공해야 한다. 기존의 웹 서비스의 메타 데이터 검색 및 조합은 서비스 소비자 중 관리자나 개발자의 역할이었다. 대다수의 일반 사용자는 직접적으로 이를 접근하고 사용하지 않았기 때문에 웹 서비스는 대중성을 얻을 수 없었다. 그러므로 대중성을 얻기 위해서는 쉽게 실행할 수 있는 인터페이

스가 필요하다. 이외에도 웹 서비스 레지스트리에서 현실적으로 필요한 요구 사항은 다음과 같이 정리될 수 있다[2].

- 존재하는 인프라 구조의 재사용
- 경량적인 접근 : 추가적인 복잡한 소프트웨어 사용 여부
- 완전한 실행 예제
- 특정 도메인에만 속한 서비스
- 사용자 알림 : 동적인 웹 서비스 변동 적용
- 다른 웹 서비스 레지스트리와의 통합
- 정확한 내용 : 서비스 제공자와 강하게 결합될수록 더 정확한 서비스
- 내용 제어 : 웹 서비스 제공자가 직접적으로 레지스트리를 제공
- 접근 및 확인이 쉬운 서비스 명세

2) 웹 서비스 프로토콜

웹 서비스는 응용 계층의 프로토콜을 사용하여 메시지를 통해 통신할 수가 있다. 웹 서비스 이용자는 HTTP 위에서 XML 기반의 SOAP과 같은 프로토콜을 이용하여 서비스 제공자로부터 서비스에 대한 요청과 응답을 수행한다. Axis나 .NET 프레임워크는 SOAP을 기본적으로 지원한다. SOAP은 XML 형태의 요청과 응답을 요구하기 때문에 응용 도메인에서 추가적인 의미(Scheme)를 쉽게 추가할 수 있다[5, 9, 11-13, 50].

[5, 9]에서는 서비스 검색 기능에 초점을 맞춘 프로토콜 정의를 보여준다. [11, 12]에서는 장치 간의 통신을 위해서 ECA(Event-Condition-Action) 개념을 따르는 WS-Eventing 기반의 프로토콜을 정의하고 이를 이용하는 프레임워크를 설계하고 구현을 했다. 실시간 응용이 필요한 경우 네트워크 계층을 기반으로 프로토콜을 정의할 수도 있다[13].

프로토콜 정의에서 문제는 기존의 웹 브라우저는 HTTP 기반의 요청/응답은 허용하지만 SOAP 요청을 직접적으로 할 수 없고, HTTP 기반의 요청 바디에 SOAP 메시지를 추가하여 요청하기 번거롭다는 점이다. 이는 서비스 인스턴스가 생성되었더라도 레거시 웹 환경에서 요청을 처리할 수 없게 만드는 요인이 된다.

웹 서비스 프로토콜에서 사용되는 데이터 타입을 사용자가 직접 정의해야 하는 것은 도메인 객체의 직렬화/역직렬화, 마샬링/언마샬링의 오래된 문제이다. XML 스키

마는 확장성이 높지만 기본 자료 구조에 대한 데이터 타입은 프레임워크 마다 다를 수밖에 없다. 새로운 프로토콜은 SOAP을 확장하고, 레거시 HTTP 요청과 응답이 가능하도록 정의되어야 한다.

3) 웹 서비스 조합

웹 서비스가 검색된 후에 단위 서비스를 가치 있는 서비스로 재창조하기 위해서는 조합 단계가 필요하다. 서비스 조합은 동적인 방법과 정적인 방법이 제안되고 있다. 동적인 방법은 실행 시점에도 서비스를 추가하거나 변경할 수 있다. 정적인 방식은 크게 두 가지 방식으로 나눌 수 있다. 첫째는 연산을 정적 선언으로부터 읽고 검색 전에 조합된 컴포넌트를 선언하고 이 컴포넌트의 인터페이스를 외부로 노출하는 방식이다. 둘째는 웹 서비스 생성 도구를 통해서 기존에 개발된 연산들을 조합하여 새로운 서비스 모델의 구조를 생성하는 방식이다. 정적인 방법은 성능이 높지만 서비스 가용성과 유연성이 낮아질 수 있다. 이와 다르게 동적인 방법은 정적인 방법에 비해 성능이 낮지만 서비스 가용성이 높고 캐시를 이용한다면 정적인 방법과 동일한 성능을 낼 수 있다. 그러므로 서비스가 로컬 서비스인 경우 정적인 조합을 사용하는 것이 좋다. 그와 반대로 원격 서비스에는 동적인 조합을 사용하는 편이 낫다.

기존 연구에서 WS-BPEL, WS-CDL과 같은 표준 언어를 통해서 어떻게 웹 서비스를 조합하고 실행하는지 보인다[14, 21, 23]. [14]에서 리플렉션을 이용한 조합을 위해서 프레임워크의 메타 모델과 BPEL 표준과의 매핑을 시도했다. [16]에서는 AOP(Aspect Oriented Programming) 방법론을 통해서 웹 서비스 기능적 변화에 대한 구현을 보인다. 서비스 바인딩 과정에서 서비스에 대한 검증이 필요하다[18]. [19]에서는 다음과 같은 기준에 따라서 서비스를 유전학 알고리즘에 따라서 조합한다.

- 실행 가격 : 서비스 요청자는 연산에 대한 비용을 지불해야 함
- 실행 시간 : 처리 시간과 전송 시간의 합
- 안정성 : 웹 서비스의 설정과 관련된 하드웨어/소프트웨어 그리고 요청자와 제공자 사이의 네트워크 연결과 관련된 기술적 측정 기준
- 가용성 : 서비스의 접근 확률
- 평판 : 시스템의 신뢰성이며 주로 종점 사용자의 경험에 의존하는 신뢰성

[17]에서는 WS-BPEL만으로 서비스를 연결할 때 에러가 쉽게 발생할 수 있으므

로 검증을 위해서 CPN(Colored Petri Net), 호환성 검사와 같은 자료 구조가 제안되었고, 이를 자동화하기 위해서 표준 혹은 전용의 언어로 매핑하는 연구가 진행 되었다[17, 20-22]. 이외에도 웹 서비스 조합에 WS-BPEL이나 WSCI를 사용할 수 있다.

서비스가 존재하는지 의미론적으로 맞는지 검증하는 것은 매우 중요하다. 검증은 올바른 동작을 보장하기 때문에 서비스의 안정성과 가용성 또한 제공하게 된다. 이러한 검증은 서비스를 사용하기 전에 메타 정보를 초기화하는 과정에서 이뤄져야 실행 시간에 성능 하락을 막을 수 있다.

4) 웹 서비스 실행

웹 서비스는 기존 분산 기술에 비해 큰 성능 하락 없이 실행될 수 있다[51]. 웹 서비스 실행 엔진은 일반적으로 요청 당 하나의 서비스 인스턴스를 생성하고 호출이 완료되면 해당 인스턴스를 소멸 혹은 캐싱을 한다. [23]에서는 웹 서비스를 조합해서 WSDL 생성 후 실행할 수 있는 도구를 제안했다. [26]에서는 표준을 따르는 자바 기반의 웹 서비스 라이브러리를 사용해서 네트워크 관리 웹 서비스 예제를 제시했고, [27]에서는 비동기 방식 웹 서비스를 요청하는 .NET 프레임워크 기반의 메일 서버와 클라이언트 사이의 웹 서비스를 구현했다.

조합된 서비스를 실행하는 중간에 사용자와 상호작용을 해야 할 필요가 있을 경우 실행 엔진이 사용자의 컨텍스트 생성 방식과 파라미터를 전달 방식을 결정하는 것이 문제가 된다. 게다가 사용자와의 상호 작용은 비지속적이라는 점이다. 그러므로 현재 오퍼레이션과 다음 오퍼레이션을 연결하기 위한 컨텍스트는 특정 퍼시스턴스 계층에 저장해야 한다. 그런 까닭에 서비스 인스턴스가 소멸되더라도 트랜잭션을 유지하기 위해서는 클라이언트를 식별할 방법, 즉 세션의 개념이 필요하다. SOAP을 이용하는 대부분의 웹 서비스 클라이언트 프록시에서 HTTP 헤더 기반의 세션 ID를 사용할 수 있다[28]. [24, 25]에서는 세션을 도입해서 Two-way 상호작용을 수행하는 방법을 보인다. 세션을 통해서 트랜잭션 처리를 하며 TARGET을 통해서 방화벽/NAT에 대한 프록시 동작을 수행하게 한다.

여러 연산 사이에서 세션이 유지되면 서비스 인스턴스는 외부 컨텍스트로부터 보안 서비스를 제공받을 수 있다. [29]에서 Choreography 선언을 통해서 명시되는 행위를 실행할 때 자원에 대한 접근 권한을 검증하는 알고리즘을 제시했다. W3C에서

는 WS-CDL, WSCI과 같은 명세를 제안하고 있다. [30]에서는 XML기반의 메시지 암호화, 디지털 시그니쳐 기능을 통합시키기 위해서 XACML(eXtensible Access Control Markup Language) 기반의 웹 서비스 관리 서버를 제안하고 서비스 제공자에게 전달된 위임 확인을 위해서 SAML(Security Assertion Markup Language)을 확장한다. 이 외에도 실행 엔진은 쉬운 개발 방법론, 프리미티브 데이터 타입 정의, 도메인 객체의 직렬화기, 실행 컨텍스트의 마샬러 등을 제공해야 한다.

5) 프레임워크

프레임워크는 서비스 등록 및 검색, 프로토콜, 조합, 실행 및 검증에 관한 전반적인 기능을 통합할 수 있는 아키텍처를 제공하는 것이 목적이다. 전체 웹 서비스의 개념을 비즈니스 로직 개발자가 구현하는 것은 생산성 하락과 많은 시간과 비용을 소모하게 된다.

동적 웹 서비스 조합에서는 WSDL을 통해서 웹 서비스의 기능을 명시하고 WSCI를 통해서 웹 서비스 사이의 상호 작용을 명시하며, Petri-Net을 통해서 웹 서비스를 검증한다[31]. ISCF는 자가 구성의 분산 웹 서비스 아키텍처를 구성한다. 이 아키텍처는 OWL-S를 비즈니스 로직 계층과 데이터 접근 계층에서 이용하고 데이터 모델을 기술하는데 ISCFTask를 온톨로지로 사용한다[32]. SASO는 사용자 레이어(시멘틱 모델링, 서비스 질의 모델, 서비스 배포자, 서비스 실행자), 조합 레이어, 실행 레이어, 서비스 레이어로 이뤄져 있다. SASO는 웹 서비스의 개념과 개념간의 관계를 설명하기 위해서 WORDNET으로부터 유도된 온톨로지를 사용한다. 서비스의 실행은 BPEL4j 라이브러리를 통해 이뤄진다[33]. Bottom-Up Approach에서 BPEL4WS를 처리하는 실행 엔진인 BPWS4J에서 발생하는 문제를 정적 조합 방식으로 꼽았다. 자동적인 사유(automatic reasoning)를 통한 서비스를 선택할 수 없기 때문에 DAML-S의 서비스 프로파일 컴포넌트를 기반으로 지식 기반(Knowledge Base)를 생성하고 DQL(DAML Query Language)를 통해서 검색할 수 있게 한다. 이는 정적 서비스 조합에서 발생하는 서비스 조합에 대한 비용을 줄일 수 있게 해준다. 이론적으로 함수적 요소(Functional Factor)에 대해서 서비스 제공자가 인지할 수 없는 상태에서 자동으로 조합을 시도하기 때문에 완전한 시멘틱이 구현되어 있어야 할 필요가 있다. 그러나 사회 문화적인 다양한 변화와 서비스 제공자와 사용자 간의 모든 도메인에 대한 이해를 적용한 완전한 시멘틱은 존재할 수 없기 때문에 현재의 접근법은

미션 크리티컬한 서비스를 제외한 실험적인 서비스에만 이용할 수 있는 수준이다[34]. METEOR-S 프레임워크에서 서비스는 제약 표현 모듈을 통해서 OWL로 표현되며, 비용 평가 모듈을 통해서 서비스의 의존성, 질의 및 비용을 평가한다. 이 때, 서비스 자체에 대한 비용으로 획득의 비용, 전달 시간, 다른 서비스 제공자와의 호환성, 관계, 서비스의 안정성, 응답 시간 등을 평가한다. 그리고 사용자는 QoS 파라미터를 위한 조합 연산자를 명시해야한다. 여기에는 실행 시간, 관련 있는 모든 서비스를 실행하는 비용, 누적된 안정성, 가용성, 도메인 특징적인 QoS 값 등이 있다. 그리고 이런 제약을 기준으로 최적화 모듈에서 최적화를 실행한다. 런타임 모듈은 추상 BPEL을 서비스 템플릿에 맞춰서 실행 BPEL을 생성하고 BPWS4J 엔진으로 넘긴다[35]. 템플릿 기반의 접근법에서 단일의 통합된 프로세스를 표현하기 위해서 사용하는 것이 템플릿이며, OWL-S를 확장하기 위해서 추상 프로세스 타입을 도입한다. Perform 구문을 통해서 추상 프로세스와 선호에 대한 정의를 조합하여 매칭과 랭킹을 시도한다. HTN-DL은 템플릿 기반의 조합 접근 방식 중 하나이다. HTN 계획 수립의 단점을 해결하기 위해 확장된 HTN-DL을 정의한다[36]. WSMO 기반의 시멘틱 웹 서비스를 생성할 수 있는 자바 및 LISP 기반의 IRS-III(Internet Reasoning Service) 프레임워크가 제안되었다. 단순한 방법으로 시멘틱 웹 서비스를 배포하고 사용자의 요구에 적합한 웹 서비스를 실행할 수 있다. 단일 웹 서비스는 WSMO의 goal을 확장한 방식으로 선언된다. IRS-III는 브라우저, WSMO 편집기, 배포 클라이언트, 실행 클라이언트를 제공한다. 이런 도구를 이용해서 정적 조합을 수행할 수 있지만 실행 중에 서비스들은 세션을 유지하지 못한다[37, 38]. SHOP2는 OWL-S 프로세스 모델을 SHOP2의 전용 도메인 모델로 변형하고, OWL-S 조합 작업을 SHOP2 계획 수립 문제로 변환하고 수립된 계획을 실행한다. SHOP2는 HTN 계획 수립 시스템이며 AI 계획 수립을 기반으로 동작한다. 상황 계산(situation calculus)에 기반을 둔 Golog와 같은 언어에서의 계획 수립은 오프라인에서 이뤄진다. 초기 상태에서 세계의 변화가 있을 수 있으므로 계획 수립 과정과 실행 계획이 일치되도록 시뮬레이션 하려면 정보를 제공하는 웹 서비스(information-providing Web Services)가 먼저 실행된 후에 실행 계획 수립이 시작되어야 한다는 점을 지적하고 있다. 또한 서비스 실행 예제에서 SHOP2 Planner는 개인 일정을 기준으로 계획을 생성하고 이를 실행하는 것을 보인다[39]. SWORD는 정적 조합을 지원하며 SOAP, WSDL, UDDI, RDF, DAML과 같은 표준에 의존하지 않는 방식으로 웹 서비스를 지

원한다. Prolog와 같은 논리형 언어에서 사용하는 규칙을 기반으로 특정 서비스에 대한 계획을 세울 수 있으며, 규칙을 통해서 질의를 수행할 수 있다. 계획은 XML을 사용해서 ER 모델을 기초로 하여 저장된다. 그렇기 때문에 서비스의 입력, 출력은 1개 이상의 엔티티로 구성될 수 있다[40]. Plaengine 프레임워크는 서비스의 시작과 목표 상태 및 진행 상태를 논리적 표현 방식으로 가지고 있는 케이스를 기준으로 조합을 수행한다. 서비스 실행 간에는 인터셉터를 통해서 실행 전, 실행 후, 예외 및 그에 대한 복원을 다룰 수 있다. 서비스의 실행이 실패될 경우 서비스에 대한 재조합을 시도하기 때문에 프레임워크의 안정성, 가용성이 높은 편이다[41]. SWS에서는 Bottom-up Approach, METEOR-S, Template Based Composition, WSMO Approach, HTN Planning using SHOP2, SWORD, Plaengine 등의 프레임워크에 대한 비교를 수행하고 있다[42]. UML과 OWL-S를 사용한 웹 서비스 진화 프레임워크에서는 유스케이스 설명을 기반으로 서비스 프로파일을 수동으로 생성한 후에 동적 진화 알고리즘에 따라서 서비스를 검색하고 찾은 서비스를 바인딩 하여 응용 프로그램을 실행한다[43].

2. 도메인 객체

웹 서비스는 하나의 도메인 아래에서 구현된다[45]. 그렇기 때문에 웹 서비스 구현에서 도메인 내의 데이터 모델링은 피해갈 수 없는 작업이다. 웹을 기반으로 하는 서비스 응용 프로그램은 데이터 모델링 후에 DBMS에 스키마를 생성하게 된다[48]. 스키마를 기반으로 SQL 구문을 작성하고 이를 처리할 퍼시스턴스 코드를 반복적으로 작성된다. 그러나 이런 반복 작업은 생산성과 성능이 낮은 시스템 구조를 양산하기 쉽다. 웹 서비스 프레임워크는 이런 반복적인 작업을 줄이고 객체와 DBMS의 분리를 제공해야 한다[38, 44]. 게다가 데이터 모델링은 또한 메타 모델의 생성, 단일 데이터의 입력, 검색, 수정, 삭제와 같은 단순 반복적인 처리를 요구한다. 이런 문제를 해결하기 위해서 SQL Map 기능을 하는 iBatis, SQL 구문 작성을 OO 모델링으로 대체한 Hibernate와 같은 ORM 프레임워크가 제안되었다. .NET 프레임워크는 DataSource를 통해서 XML Schema를 생성하고 DataSet에 데이터 값을 채워주는 DataFillAdapter를 지원한다. 이러한 ORM 프레임워크는 특정 모델에 대한 복잡한 동작까지 예측하여 처리하지는 않는다. 그러므로 복잡한 퍼시스턴스 연산은 수동으

로 구현해야 한다.

프레임워크는 객체와 퍼시스턴스 계층 간의 매핑 대신 객체와 XML을 매핑해 줄 필요가 있다. OR 매핑에서 발생되는 클래스 정의는 반복적인데다가 클래스는 단순히 데이터 전송 객체(Data Transfer Object)의 역할만 수행하기 때문에 생산성을 낮추는 가장 커다란 요인 중 하나가 된다. 웹 서비스 환경에서는 메시지의 기본 단위가 XML이기 때문에 객체 상태는 XML 문서로 표현되어야 한다. 이를 위해서는 도메인 객체에 대한 인터페이스가 필요하다.

[48]에서는 XML을 통해서 표현된 데이터를 객체 지향 언어의 클래스에 매핑하는 방법을 제안한다. 먼저 XElement 방식은 DOM과 비슷한 방식으로 XML을 다루지만 사용자는 XML 파싱에서 사용되는 핸들러 객체에 특정 XML 요소에 대해서 클래스를 등록한다. 핸들러 객체에서 해당 요소를 파싱할 때 등록된 클래스를 초기화하고 그 객체에 값을 할당한다. 핸들러 객체에 등록되는 객체는 반드시 XElement 클래스를 상속해야 한다.

다음으로 NaturalXML 방식은 메타 데이터를 각 클래스에 추가한다. 메타 데이터는 XML과 매핑하려는 클래스의 필드와의 관계를 명시한다. 이 방식은 순서를 유지못한다는 단점이 존재한다. 데이터는 XIR이라는 Base64 인코딩을 지원하는 방식으로 전달될 수 있다. 다만, 기존에 존재하는 XML이나 JSON 보다 오히려 더 장황하고 복잡하다.

도메인 객체는 서비스에서 사용하는 모델과 뷰를 나타내는 인스턴스이다. 뷰(View) 계층부터 퍼시스턴스 계층까지 도메인 오브젝트는 다양한 형태로 선언되고 사용된다. 도메인 객체는 각 레이어 사이에서 통신을 할 때 해당 레이어에 알맞은 형태로 변형되어야 이용이 가능하다. 모든 도메인 객체는 XML 형태를 유지하지 않으며, 억지로 유지하더라도 성능 상의 문제를 발생시킨다. 그러나 웹 서비스 프레임워크에서 사용되는 도메인 객체는 성능보다도 서비스 간의 원활한 메시지 전달을 위해서 XML 형태를 가지고 있어야 한다. 그러므로 프리미티브 타입에서 Object 타입까지 XML 형태로 변환할 수 있는 기능이 제공되어야 한다.

[45]에서는 Naked Objects 프레임워크가 제안되었다. 이 프레임워크는 동적으로 서비스를 추가하고 사용할 수 있다. 서비스가 변경되면 GUI와 퍼시스턴스 계층에 변경된 내용이 반영되기 때문에 서비스 모델링이 자동적으로 된다. [46-48]에서는 서비스 코드와 데이터를 다루는 코드를 분리하기 위한 아이디어를 제안되었다.

III. DOKDO-WS 프레임워크

1. 전체 프레임워크 구조

제안하는 프레임워크는 그림 1과 같이 크게 4개의 레이어로 구성된다. 레이어는 MVC 패턴을 기반으로 분할되어 있다. 뷰 레이어는 일반 서비스 사용자에게 UI를 제공하며, 실행 레이어는 처리 흐름 및 데이터 전달 기능을 담당한다. 서비스 모델과 데이터 모델 레이어는 시스템 전체에서 사용되는 도메인 모델을 제공한다.

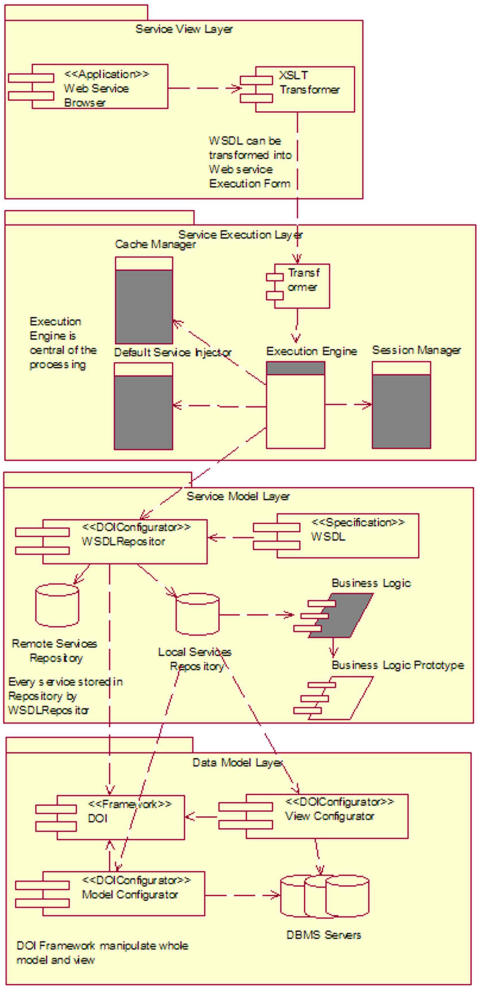

그림 1. 전체 프레임워크 구조(다이어그램)

1) 데이터 모델 계층

이 계층은 비즈니스 로직에서 사용되는 데이터와 서비스 사용자에게 제시할 뷰를 위한 모델링을 다루기 위한 서비스를 제공하는 계층이다. 대부분의 작업은 Domain Object Interface(DOI) 프레임워크에게 위임하여 이뤄진다. DOI는 XML 기반의 도메인 객체에 대한 로딩, 검색, 수정, 저장 작업을 위한 인터페이스를 담고 있다. 이런 인터페이스를 이용하여 서비스 제공자는 서비스에 대한 모델과 뷰에 대한 도메인 객체 설정을 하고 DOKDO-WS 프레임워크는 DOI를 이용하여 비즈니스 로직의 도메인 모델을 생성한다.

2) 서비스 모델 계층

이 계층은 원격의 WSDL 문서에 접근하여 서비스가 사용 가능한지 검증을 하고 사용할 수 있을 경우 런타임 저장소에 추가한다. 또한 로컬의 서비스에 대해서 비즈니스 로직의 프로토타입을 비즈니스 로직 컴포넌트들로부터 획득하고 이를 기준으로 WSDL 객체를 생성하여 프레임워크 초기화 시점에 같이 생성한다. 프레임워크는 검증된 서비스만을 런타임 저장소에 추가하기 때문에 서비스 인스턴스는 반드시 실행이 보장된다.

3) 서비스 실행 계층

이 계층에서는 실행 요청이 들어 왔을 때 이를 처리하는 방식에 대한 정책과 전략을 결정하게 된다. 동기식, 비동기식 실행, 세션 유지 여부, 캐시 정책, 그리고 인스턴스 획득 방식 등이 선택 사항이다. 서비스 실행 엔진 하위에는 캐쉬 관리자, 세션 관리자, 서비스 주입자 서비스가 존재한다.

(1) 캐쉬 관리자

특정 서비스 연산은 반복적으로 짧은 시간에 호출될 수 있기 때문에 호출 당 인스턴스를 생성하는 것은 서비스의 성능을 매우 낮출 수 있다. 그렇기 때문에 캐시 관리자는 자주 사용되는 서비스에 대해서는 일정 기간 동안 인스턴스를 캐시해 둔다.

(2) 세션 관리자

특정 조합의 서비스에서 세션 개념이 필요할 수가 있다. 도메인 객체 안의 데이터 중 상태 유지가 필요한 데이터는 서비스 선언 시에 선택할 수 있어야 한다. 세션이

가능하도록 선택되었다면 세션 관리자는 호출 전에 이전 세션의 값을 불러오고, 서비스를 호출한 후에 도메인 객체 내의 새 값을 세션 저장소에 다시 저장해야 한다.

(3) 서비스 주입자

서비스 주입자는 특정 웹 서비스 인스턴스가 기본적으로 제공해야 하는 서비스들에 대한 주입을 수행한다. 이는 실행 계층에서 서비스에 대한 동적 조합을 가능하게 한다. 기본적으로 주입되는 서비스는 세션 관리자이며, 서비스 인스턴스에 세션 처리를 가능하도록 한다.

4) 서비스 뷰 계층

서비스 뷰 계층에 있는 도메인 객체는 XML 기반의 메시지를 받게 된다. 그러나 일반적으로 웹 서비스 제공자는 서비스 실행 레이어에서 넘겨진 결과를 GUI로 가공한다. 이를 위해서 프레임워크는 XSLT를 이용하여 사용자가 원하는 뷰 모델을 생성한다. 또한 서비스 제공자가 이용할 수 있는 웹 서비스 브라우징 서비스와 브라우징한 연산을 실행할 수 있는 서비스를 제공한다.

2. 프레임워크 설계

1) 유스 케이스 설명

그림 2에서는 서비스 제공 개발자의 유스케이스를 보인다. 서비스 제공자 개발자는 제공할 서비스를 위한 모델과 뷰에 대한 설정을 선언하고 이를 기준으로 비즈니스 로직을 나타내는 컨트롤러 클래스를 개발한다. 다음에 개발된 컨트롤러 클래스에 대한 설정을 서비스 모델에 추가해야 한다. 컨트롤러 클래스를 개발할 때는 해당 서비스의 각 연산 혹은 서비스 전체가 외부로 노출될 것인지 여부와 세션을 사용할 것인지 여부를 명시해야 한다.

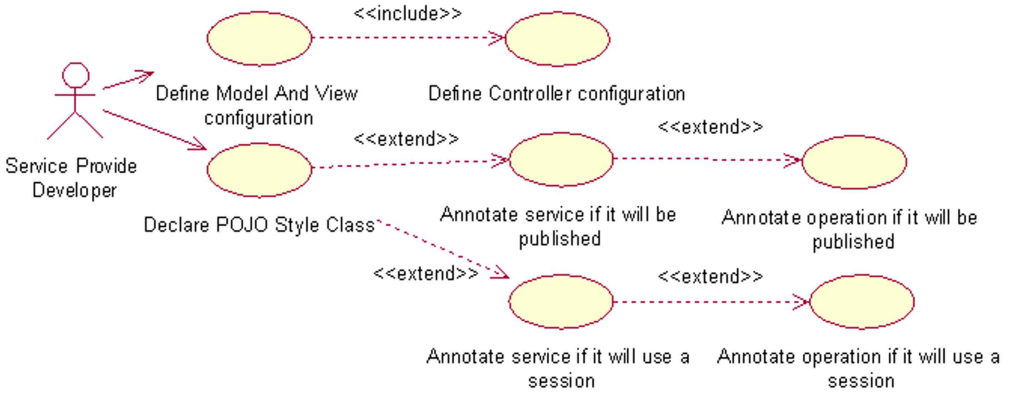

그림 2. 서비스 제공 개발자

그림 3에서는 서비스 조합 개발자의 유스케이스를 보인다. 서비스 조합 개발자는 메타 서비스를 통해서 연산을 확인하고 연산을 나타내는 WSDL을 기반으로 프록시를 생성한다. 또한 생성된 프록시에서 연산을 조합하여 가치 있는 서비스를 새롭게 개발한다.

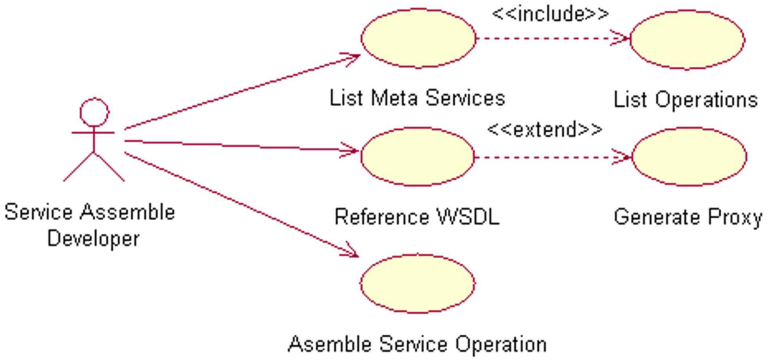

그림 3. 서비스 조합 개발자

그림 4에서 DOKDO-WS 프레임워크의 유스케이스를 보인다. DOKDO-WS 프레임워크는 서비스 제공 개발자와 조합 개발자가 정의한 설정 파일에 따라서 컨트롤러 역할을 하는 로컬 및 원격 서비스를 검증 후 생성한다. 이 과정은 WSDL 객체와 서비스 모델 생성 과정을 포함한다. 다음으로 모델과 뷰를 퍼시스턴스 계층의 RDBMS에 생성한다. 이 때 서비스에서 퍼시스턴스 연산 로직으로 사용할 저장 프로시저를 자동으로 생성한다. 퍼시스턴스 연산 로직에는 삽입, 검색 가능한 열에 대한 검색, 수정, 삭제 연산을 포함한다.

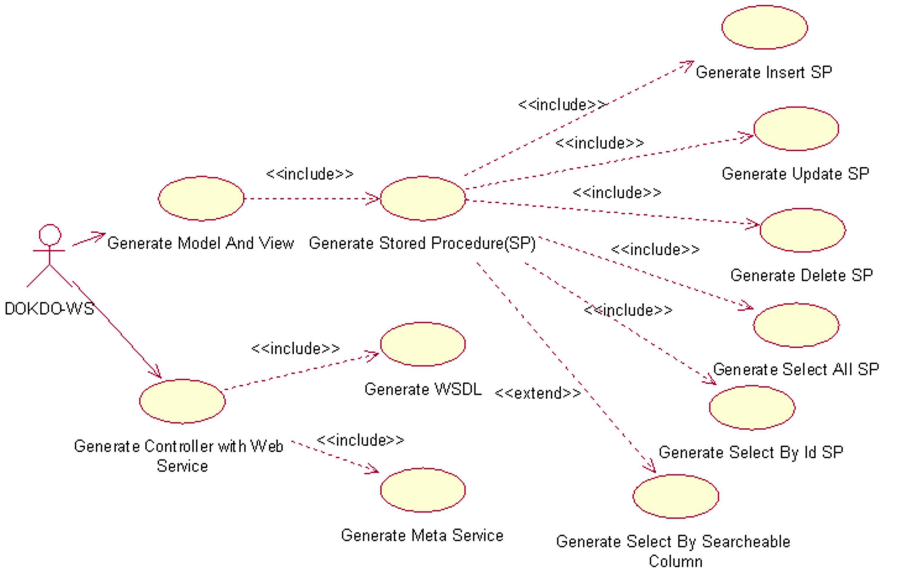

그림 4. DOKDO-WS 프레임워크

2) 상태 다이어그램

5는 컨테이너 초기화 단계에 대한 상태 다이어그램을 보인다. 처음 상태인 컨텍스트 초기화 상태에서 다른 프레임워크도 초기화될 수 있다. 다음으로 컨트롤러 생성 상태에서 실행 엔진이 초기화 된다. 서비스 로드 상태에서 서비스 선언을 서비스 모델로 변환한다. WSDL 생성 상태에서 WSDL 객체를 서비스 모델 별로 생성한다. 다음 상태는 서비스 모델의 설정에 따라서 결정되며 모델과 뷰를 생성하는 상태를 거치거나 거치지 않게 된다. 최종적으로 DOI에 대한 컨텍스트의 생성 완료 상태가 된다.

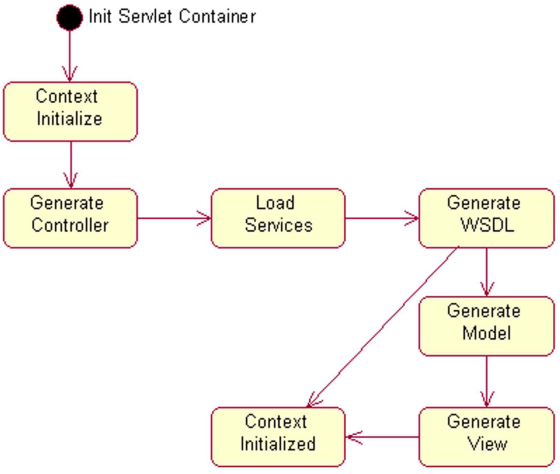

그림 5. 컨테이너 초기화

그림 6은 웹 서비스 실행에 대한 상태 다이어그램을 보인다. 그림 5의 상태 전이를 통해서 컨텍스트 생성이 완료된 후에 외부의 클라이언트는 서비스 나열 상태에 있게 된다. 서비스들은 이름 공간에 의해서 그룹 별로 묶인 상태로 보인다. WSDL 질의 상태에서 WSDL 객체들에서 WSDL 문서를 얻을 수 있다. 연산 리스트 상태에서 클라이언트는 사용 가능한 원격 및 로컬 서비스의 연산들을 나열할 수 있다. 다음 상태에서 연산에 파라미터를 채우고 난 후에 서비스 실행 엔진은 서비스들을 실행할 수 있다. 최종 상태에서 웹 서비스 실행 결과를 표시하게 된다.

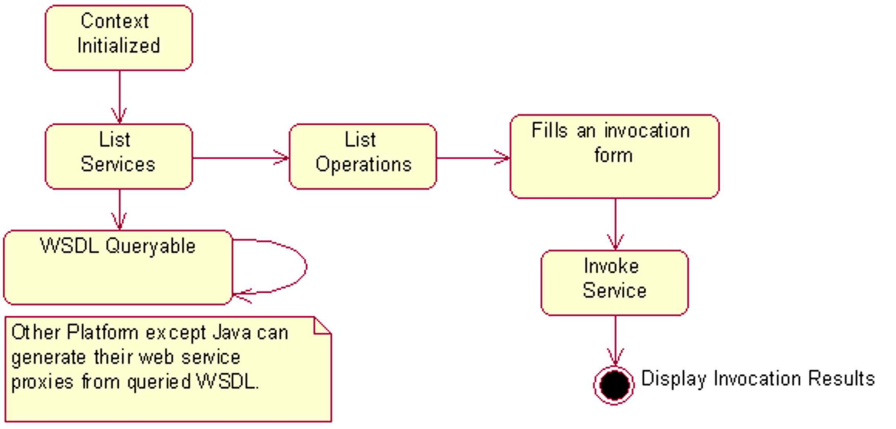

그림 6. 웹 서비스 실행

3) 컴포넌트 설계

(1) 비즈니스 로직 계층 설계

① DOI

그림 7은 DOI에 대한 클래스 다이어그램이다. IDomainObject 인터페이스는 모든 도메인 객체를 위한 인터페이스이며 IDOMAccessor와 IDataAccess Object를 확장한다. IDOMAccessor 인터페이스는 설정에서부터 실행에 이르기까지 사용되는 XML 기반의 모든 도메인 객체를 조작하기 위한 인터페이스이다. IDataAccessObject 인터페이스는 도메인 객체를 퍼시스턴스 계층에서 로딩, 수정, 삭제하기 위한 인터페이스이다. IDomainObject에 대한 기본 구현은 AbDomainObject 추상 클래스를 통해서 제공된다. 이 추상 클래스에서는 XML로 표현된 도메인 객체의 현재 상태를 DOM을 통해 다루기 위한 Document 객체와 XML 문서에서 노드를 검색하기 위한 XPath 객체를 생성한다.

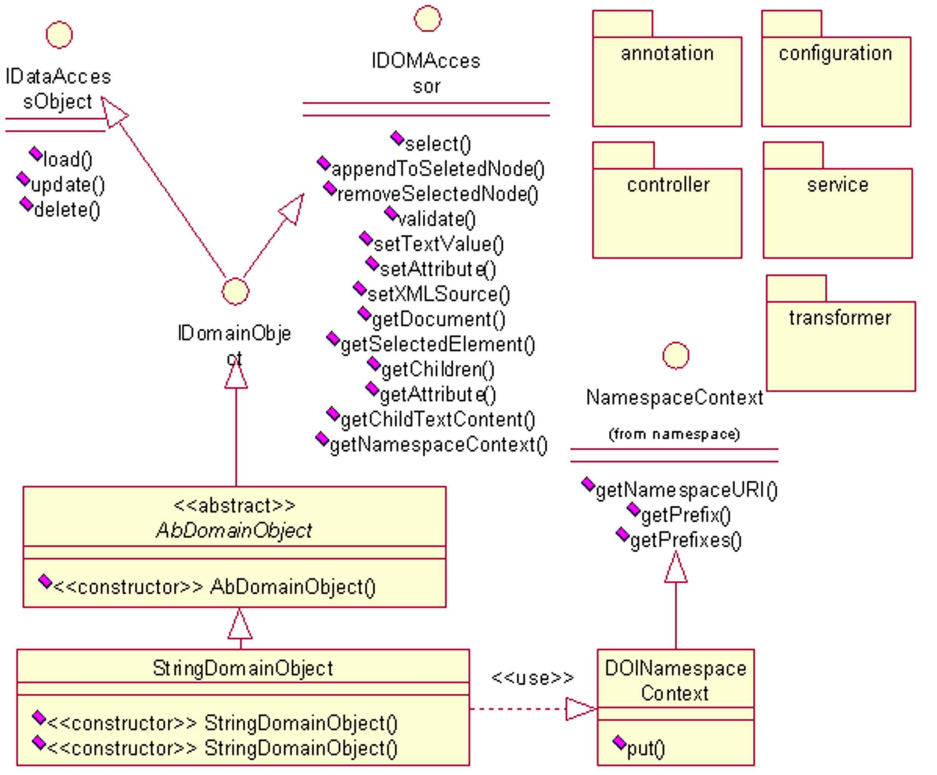

그림 7. DOI 클래스 다이어그램

② 데이터베이스 퍼사드

그림 8은 DBMS에 접근할 때 사용하기 위한 데이터 베이스 퍼사드를 위한 인터페이스에 대한 클래스 다이어그램이다. IDatabaseFacade 인터페이스는 모든 데이터베이스 접근 객체가 반드시 제공해야하는 기능을 캡슐화한 인터페이스이다. ICloseable 인터페이스는 해당 객체가 닫힐 수 있음을 나타내지만, 객체 사용자는 이 메서드를 호출할 필요가 없다. 왜냐하면 이 인터페이스는 Template Method 패턴을 구현하기 때문에 DBMS 연결 해제를 자동으로 수행하기 때문이다. AbDatabaseFacade 클래스는 IDatabaseFacade 인터페이스에 대한 기본 구현을 제공하는 추상 클래스이다. AbJDBCDatabaseFacade 추상 클래스는 많은 데이터베이스 연결 중 특히 자바의 JDBC 인터페이스를 사용하는 객체를 위한 기본 구현을 제공하는 추상 클래스이다. 또한 사용의 편의를 위해서 프레임워크는 자주 사용되는 Oracle, MS-SQL, MySQL DBMS를 위한 구현클래스를 제공한다.

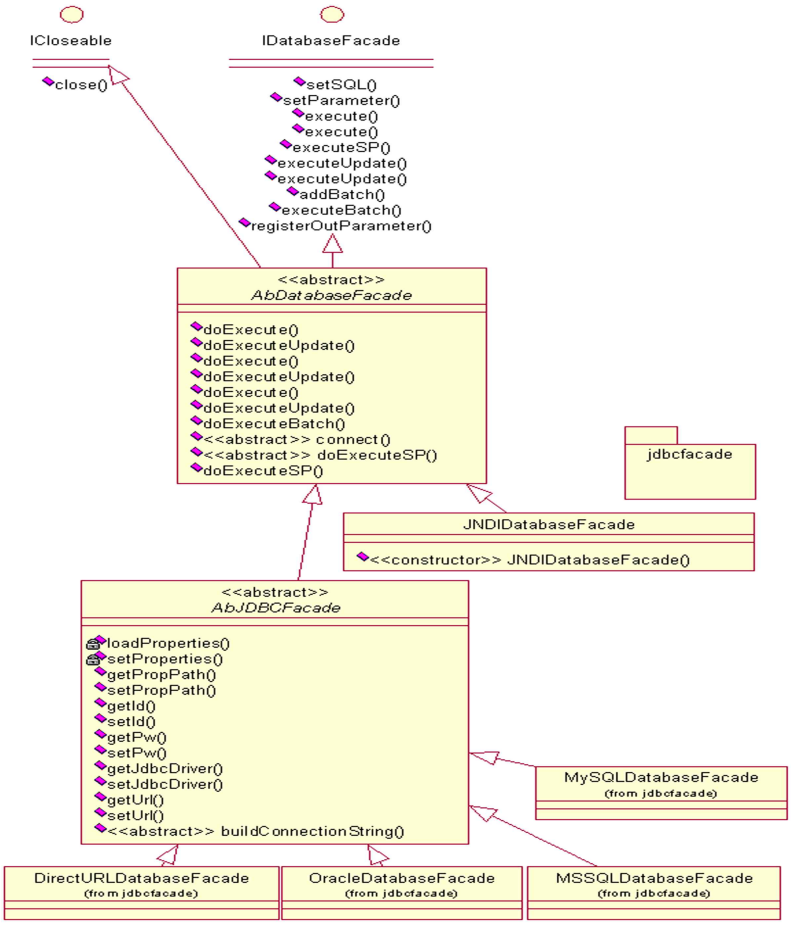

그림 8. 데이터베이스 퍼사드 클래스 다이어그램

③ 설정자

그림 9는 이 프레임워크에서 서비스와 데이터 모델에 대한 설정을 읽어서 런타임에 사용하기 위한 설정자의 클래스 다이어그램을 보인다. 프레임워크에서 사용하는 모든 설정을 다루기 위해서 DOI의 기본 구현을 제공하는 AbDomainObject 클래스를 상속하는 AbDOIConfigurator 추상 클래스를 정의한다. DBMS에 대한 설정을 위해서 AbDatabaseConfigurator 추상 클래스를 정의하고 이를 확장하여 모델과 뷰에 대한 설정자를 위한 추상 클래스를 정의한다. 최종적으로 서비스 인스턴스로 생성될 수 있는 구체 설정자를 정의한다. 서비스 실행 엔진이 배포되는 곳에 따라서 초기화를 수행할 수 있는 설정자 클래스를 정의해야 하며 자바 웹 환경을 위한 DOIServletConfigurator 구체 클래스를 제공한다.

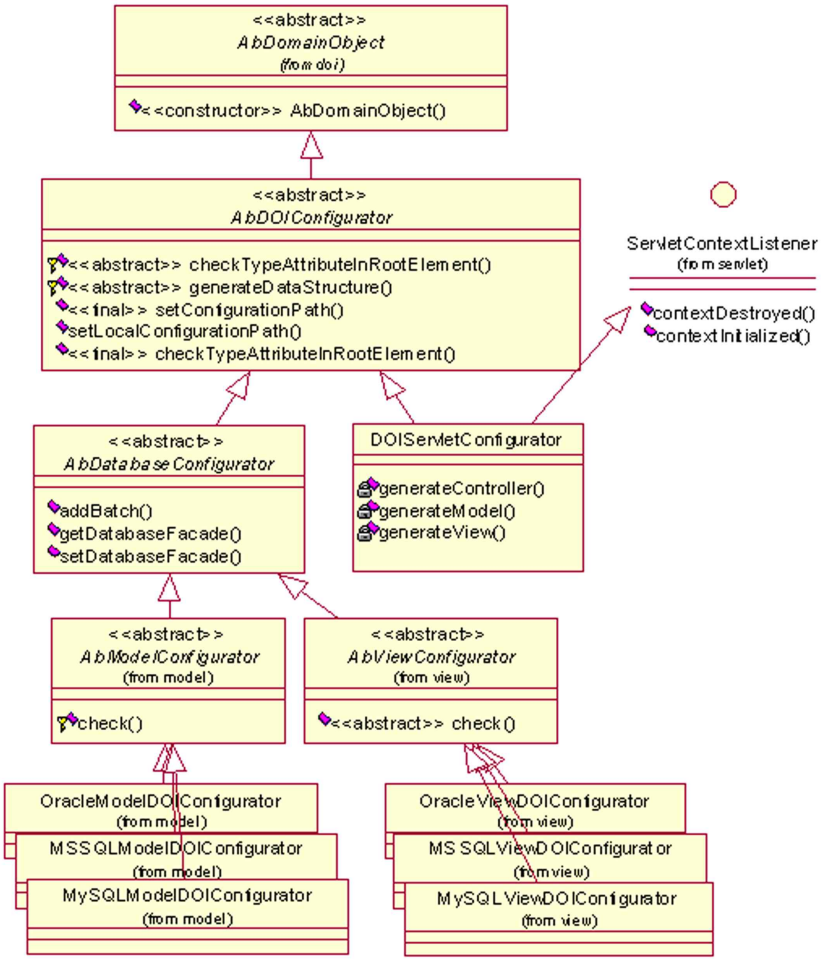

그림 9. 설정자 클래스 다이어그램

(2) 서비스 메타 모델 계층과 서비스 실행 계층의 설계

그림 10은 서비스 모델의 저장소 클래스와 여기에 저장되는 메타 데이터를 다루기 위한 실행기 클래스에 대한 클래스 다이어그램이다. 서비스 저장소는 차후에 웹 서비스 실행을 위해서 WSDL 문서를 저장 혹은 생성하고 서비스를 나열하거나 질의하며, 서비스 별로 오퍼레이션 나열을 할 수 있는 인터페이스를 제공한다.

실행기는 서비스를 실행하기 위한 목적을 가지며 하위 프로그램인 캐쉬 관리자, 세션 관리자, 서비스 주입자의 기능을 구현한다. 로컬과 원격 서비스 처리를 위한 실행기가 따로 있으며, 기본적으로 모든 데이터는 XML 형태를 가지기 때문에 모든 실행자는 DOIXMLExecutor에서 파생되도록 구현한다.

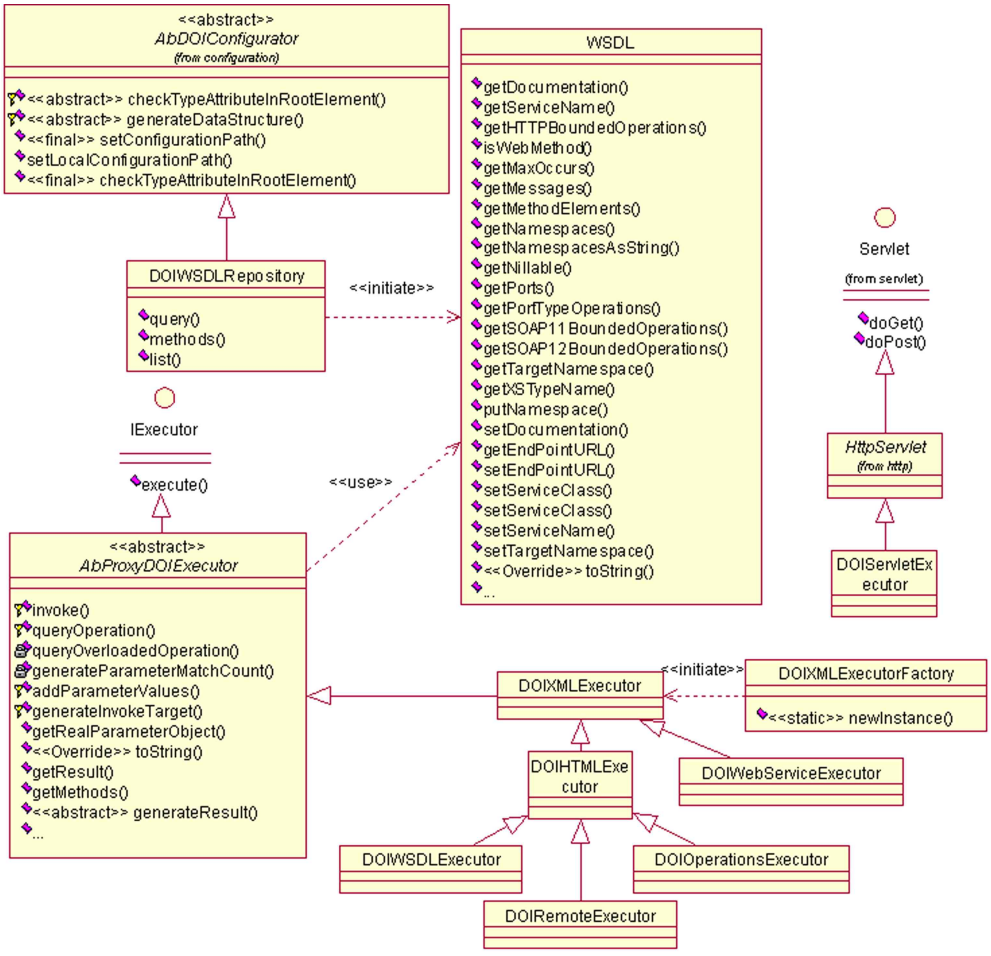

그림 10. WSDL 저장소와 실행기 클래스 다이어그램

그림 11은 서비스 뷰 계층으로 전달하기 전에 도메인 객체 내의 데이터를 XML 형태로 변환하기 위해서 사용하는 변환기에 대한 클래스 다이어그램이다. 서비스가 현재 실행 컨텍스트 외부의 데이터를 메시지에 사용할 때는 TypeSerializer 서비스를 통해서 도메인 객체로 변환하고 반대로 실행 엔진이 도메인 객체를 다른 계층으로 전달할 때는 문자열, XML 문서 형태로 변환된다.

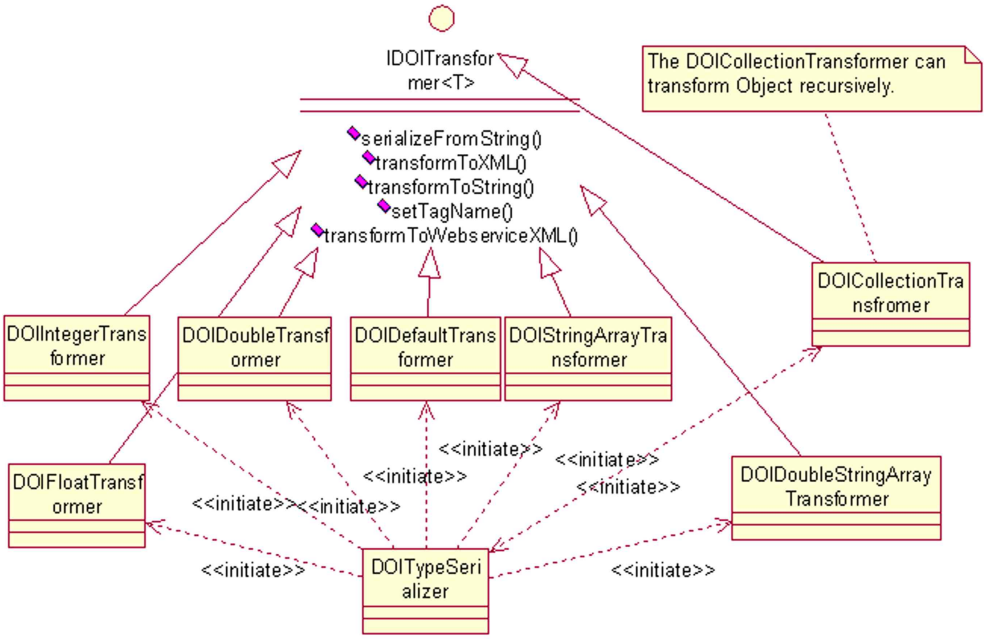

그림 11. 변환기 클래스 다이어그램

(3) 서비스 뷰 계층의 설계

그림 12는 서비스 뷰 계층에서 XML 문서를 다른 형태의 XML 문서로 변환하기 위해서 사용하는 XSLT 처리기에 대한 클래스 다이어그램이다. 이 서비스를 사용하여 서비스 실행 계층에서 반환된 XML 문서를 서비스 뷰 계층에 알맞은 형태로 변환시킬 수 있다. 만약 실행 계층에서 단순히 문자 기반의 스트림이 반환되었다면 뷰 계층에서는 XSLT 변환을 사용할 필요가 없고, 스트림을 인식하여 처리하는 클라이언트 응용 프로그램이 존재해야 한다.

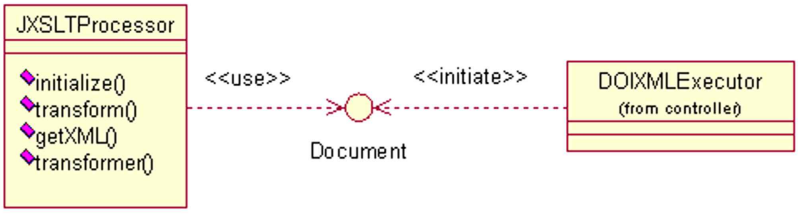

그림 12. XSLT 처리기 클래스 다이어그램

4) 메타 모델 설계

(1) 중심 메타 모델

DOKDO-WS 프레임워크에서 사용하는 모든 도메인 객체의 설정에 대한 시작은 그림 13의 중심 메타 모델에서 시작한다. 이 모델은 XML 스키마로 선언된다. 모든 메타 모델 인스턴스는 루트 노드의 type 속성에 반드시 해당 모델의 타입 값을 가진다. 이 속성은 서비스 제공자가 특정 메타 모델 인스턴스의 타입을 정확히 알고 메타 모델을 선언한 것을 검증하기 위하여 존재한다. modelAndViewGenerate 속성은 그림 5에서 Generate WSDL 상태 다음에 상태를 결정하는 속성이다. 만약 true의 값을 가진다면 데이터 모델 계층을 생성하게 된다. 서비스 제공자는 한 도메인에 오직 하나의 중심 메타 모델을 선언해야 한다. 왜냐하면 로컬 서비스 저장소는 모든 서비스의 시작점을 동일하게 유지하고 해싱 기반의 서비스 검색을 수행하기 때문이다. 모든 모델에서 localPath, remotePath 속성은 도메인 객체의 입력/출력 위치를 나타내는 경로이며, id 속성은 모델 정의에서 개별 모델을 식별하는 식별자이다.

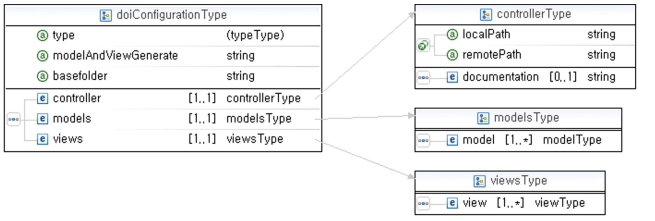

그림 13. 중심 메타 모델

그림 14는 도메인 모델과 뷰에 대한 모델을 보인다. 도메인 데이터 모델과 도메인 뷰 모델은 유사한 선언을 가진다. 도메인 데이터 모델은 웹 서비스의 도메인에서 사용되는 DBMS 기반 퍼시스턴스의 모든 정보를 설명하는 모델이다. dbmsName 속성은 프레임워크에서 지원하는 DBMS 이름만을 값으로 가질 수 있다. diName 속성은 디렉토리 서비스로부터 디렉토리 인터페이스를 통해서 특정 자원을 찾을 때 사용하는 이름을 명시하는데 사용된다. 프레임워크의 참조 구현이 자바를 이용하므로 속성 이름은 jndiName이다.

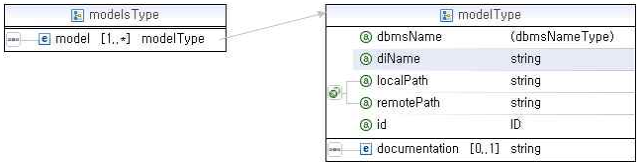

그림 14. 도메인 데이터 모델과 도메인 뷰 모델

그림 15는 중심 메타 모델의 인스턴스의 예이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<doicfg:doi    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:doi="http://javawide.com/DOIconfiguration" xsi:schemaLocation=
"http://javawide.com/DOIconfigration..." type="configuration" model
AndViewGenerate="true">
<doicfg:controllers>
<doicfg:controller localPath="protocol:///path/filename" />
...
</doicfg:controllers>
<doicfg:models>
<doicfg:model  localPath="protocol:///path/filename"  dbmsName="Some
DBMS" jndiName="jdbc/doi_somedbms" />
...
<doicfg:views>
<doicfg:view  remotePath="protocol:///path/filename"  dbmsName="Some
DBMS" jndiName="jdbc/doi_somedbms" />
...
</doicfg:views>
</doicfg:doi>
```

그림 15. 중심 메타 모델의 인스턴스

(2) 하위 구체 모델

① 서비스 모델

그림 16은 서비스 모델을 보인다. 서비스 모델은 단일 프로세스들을 표현하기 위한 모델로 MVC 패턴의 컨트롤러로 생각할 수 있다. 모든 비즈니스 로직은 로컬과 원격 서비스에 관련 없이 동일한 형태를 가진다. 공통적으로 사용하는 노드는 다음과 같다. controllers 요소의 id 속성은 서비스의 이름 공간 역할을 하며 controller 요소의 name 속성은 서비스들의 유일한 식별자다. documentation 요소는 서비스를 설

명하며, targetNamespace 요소는 호출자의 이름 공간을 나타낸다.

serviceClass 요소는 로컬 서비스에서만 사용되며 서비스 인스턴스를 생성하기 위해서 사용하는 모듈 이름을 나타낸다. 그리고 endPointURL은 실행 종점 주소를 나타내며 외부에서 SOAP 요청을 받는 URL이다. 마지막으로 원격 서비스에서만 사용되는 요소를 살펴보면, wsdlURL은 실행하려는 서비스 명세의 URL이다.

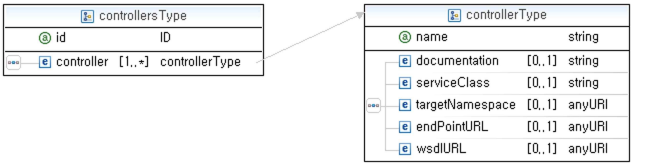

그림 16. 구체 서비스 모델

② 데이터 모델

그림 17은 구체 데이터 모델을 보인다. 데이터 모델은 단일 프로세스에서 사용되는 객체와 데이터를 표현하기 위한 모델로 MVC 패턴의 모델과 뷰를 나타낸다. DOKDO-WS 프레임워크는 기본적으로 데이터를 RDBMS에 저장하기 때문에 데이터 모델은 ER 모델의 형태를 가지게 된다. 데이터 모델은 database 요소 아래에 다수의 table 혹은 view 요소를 가지고 있다. 프레임워크는 구체 데이터 모델을 기준으로 DBMS 상에 도메인 데이터 모델을 생성한다. forceRemove 속성이 true일 경우에 해당 모델은 강제적으로 삭제되고 다시 생성된다. primaryKey 요소는 column 요소와 동일한 타입을 가지지만 ER 모델의 주키를 표현한다. 선언에서 복합 키는 허용하지 않는데 설정자 서비스는 내부적으로 id라는 속성으로 인공키를 생성하고 사용자가 하나의 주 키를 선언하게 되면 id와 주 키를 합쳐서 유일키를 생성할 수 있기 때문이다. columnType 안에 있는 속성은 퍼시스턴스 값을 도메인 객체로의 매핑을 수행할 때 제약 조건이나 데이터 타입 및 이름을 담게 된다.

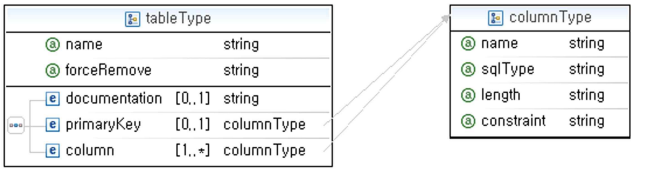

그림 17. 구체 데이터 모델

그림 18은 서비스 뷰 계층에서 사용하는 구체 뷰 모델을 보인다. force Remove

속성은 데이터 모델과 마찬가지로 생성한 뷰 모델을 강제적으로 지우고 재생성하는 것을 나타낸다. mapTo 속성은 뷰 계층에 노출되는 데이터의 이름을 수정할 수 있다. generate 속성은 저장 프로시저로 생성할 퍼시스턴스 연산을 명시한다. 기본적으로 수정, 삭제 연산은 내부적인 id 열에 대해서만 생성되지만 이 속성은 다른 열에 대해서도 생성할 수 있도록 허용하게 해준다.

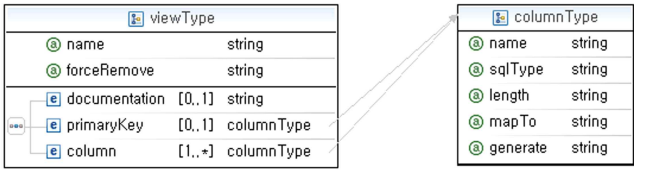

그림 18. 구체 뷰 모델

IV. 구현 및 성능평가

1. 구현 및 성능평가 환경

본 논문의 성능평가에서 CPU는 AMD Athlon 64 bit Dual Core 2.0Ghz를 사용했고, OS는 윈도우즈 XP Professional SP3를 사용했으며, 프로그래밍 환경으로 JDK 1.6.0 Update 10 버전을 이용한다. 메모리는 2.75GB의 물리 메모리, 가상 메모리 관점에서 전체 3.6GB의 환경에서 네트워크 타입은 유선, 지리적 위치는 로컬, 프로토콜은 HTTP를 이용해서 평가 한다. 특정 서비스의 요청은 HttpClient 4.0 Beta 1 버전을 사용한다.

성능 평가는 웹 서비스와 도메인 객체 별로 정량적 평가와 정성적 평가를 수행한다. 웹 서비스의 정량적 성능 평가에서 요청 횟수는 5000번, 강도는 30 threads/sec 정도로 평가했으며 평가 기준은 다른 조건은 모두 동일하기 때문에 서버 실행 시간만을 이용한다.

도메인 객체의 정량적 성능 평가에서 요청 횟수는 10만 번, 강도는 30 threads/sec로 은행 응용 프로그램에서 평가한다. 다만, JOXM의 경우 30 threads/sec에서 실패율이 100%였기 때문에 요청 횟수는 2만번, 강도는 10 threads/sec로 조정하여 평가 한다. 변경되는 평가 항목은 사용하는 퍼시스턴스 종류, 사용한 퍼시스턴스 연산의 종류, ORM 프레임워크의 종류이다. 평가 기준은 서버 실행 시간, 클라이언트 수신 시간, 네트워크 소요 시간이다. 클라이언트 수신 시간에서 서버 실행 시간을 빼서 네트워크 소요 시간을 계산한다. 퍼시스턴스 종류는 JOXM, MS-SQL, Oracle, MySQL이며 ORM 프레임워크에서 iBatis, Hibernate를 이용하여 비교한다. 퍼시스턴스 연산의 종류는 입력, 수정, 삭제, 조합, 선택의 연산이 있다. 입력, 수정, 삭제, 조합 연산에 대해서는 서버 실행 시간을 평가 기준으로 사용했고, 선택 연산은 서버 실행 시간과 전체 클라이언트 수신 시간, 네트워크 소요 시간을 평가 기준으로 사용한다. 예외적으로 서버 실행 시간이 500ms 이상일 경우 가비지 컬렉션이 발생한 것으로 추측하고 해당 값은 제외한다.

2. 성능평가 결과

1) 웹 서비스 측면

(1) 정성적 평가

표 1과 표 2는 기존의 웹 서비스 프레임워크와 DOKDO-WS의 특징을 비교하고 있다. 메타 데이터 저장소로 UDDI를 이용하는 것은 표준 웹 서비스 방식을 사용하는 것이다. 시멘틱 웹 서비스를 제공하기 위해서는 독자적인 메타 데이터 저장소를 제공할 필요가 있다. 중앙 집중 저장 방식은 전역적인 서비스 레지스트리에서 서비스 실행 명세를 질의할 수 있는 방식이므로 분산 방식이 웹 서비스에 더 적합하다. 경량적인 접근은 프레임워크가 추가적인 도구가 필요하지 않도록 개발 프로세스를 자동화하거나 기존의 개발 방법론을 유지하는 등의 요건을 만족시키는 것을 의미한다. 기존 연구 중 동적 웹 서비스 조합, UML 기반 진화 프레임워크만이 경량적인 접근법을 제공한다. 메타 모델 언어는 다른 웹 서비스 프레임워크의 도메인 모델과 호환될 수 있는지를 나타낼 수 있다. 이를 위해서는 표준 도메인 모델링 언어를 사용하거나 일반적인 도메인 모델을 다룰 수 있는 인터페이스를 제공해야 한다. SWORD가 이런 인터페이스를 제공했다. 조합 방식은 프레임워크 성능과 편리성 및 확장성, 학습 주기에 큰 영향을 미친다. 대부분의 기존 프레임워크에서의 웹 서비스 조합은 논리형 언어의 규칙을 이용하여 지식 기반을 생성하고 이를 통해 웹 서비스를 검색하고 조합한다. 그러나 논리형 언어의 구문에 대한 추가적인 학습 시간과 표준 언어와 통합하기 위해서 매핑 도구나 API를 요구하게 된다. AI 방식은 최소의 설정으로 조합하지만 성능이 낮다. Static 방식이 가장 사용하기 쉽고 확장성이 높지만 서비스 제공자가 일일이 모든 설정을 해야 하는 단점이 있다. 실행 명세는 실행할 때 사용하는 서비스와 연산을 설명하는 언어이다. WSDL, BPEL의 표준을 이용하는 방법이 더 높은 확장성을 가진다.

표 1. 웹 서비스 프레임워크와의 특징 비교 - I

항목	동적 웹 서비스 조합	ISCF	SASO	Bottom-Up Approach	METEOR-S	Template-based
메타 데이터저장소	UDDI	OWL-S Library	SDL	DQL Server	UDDI	HTN
중앙 집중 저장	○	○	×	×	○	○
경량적 접근	○	×	×	×	×	×
메타 모델 언어	WSCI	OWL-S	WORD NET	DAML-S	RosettaNet	OWL-S
조합 방식	Rule Base	Rule base	Rule base	Static	Rule base	HTN
실행 명세	BPEL	WSDL/ICL	BPEL, WSDL	BPEL4WS	BPEL, WSDL	HTN-DL

표 2. 웹 서비스 프레임워크와의 특징 비교 - II

항목	IRS-III	SHOP2	SWORD	Plaengine	SWS	UML 기반 진화	DOKDO-WS
메타 데이터 저장소	OCML Library	HTN	Persistent XML Plan	Domain Registry	SWSs registry	Ontology Repository	Domain Registry
중앙 집중 저장	○	○	○	○	×	○	×
경량적 접근	×	×	×	×	×	○	○
기본 메타 모델 언어	WSMO, UPML	SHOP2 Domain(from OWL-S)	Every XML	Own Meta-Model	OWL-S	OWL-S, OWL	Every XML
조합 방식	Rule base	HTN	Rule base	AI	AI	AI	Dynamic
실행 명세	WSMO	WSDL	WebL, Filter	WS-BPEL	BPEL4WS	WSDL	WSDL, Function Structure

(2) 정량적 평가

10 만 번의 퍼시스턴스 연산 실행 과정에서 에러는 평균 0~3개 정도 발생한다. 그

림 18에서 DOKDO-WS의 서비스 레지스트리 연산의 성능을 보인다. 서비스 등록 연산은 연산 시간이 점점 증가한다. 이 연산은 서비스 메타 데이터가 물리 디스크에 쓰일 때만 많은 시간이 소요된다. 로컬 서비스가 원격 서비스 보다 시간의 기울기가 더 큰 것은 로컬 서비스는 추가되는 서비스 모듈마다 인스턴스화를 시도함으로써 존재 여부를 검증하기 때문이다. 전반적으로 서비스 수정은 가장 많은 시간이 걸린다. 왜냐하면 이 연산에는 서비스의 위치를 검색하고 수정하는 두 개의 연산이 합쳐져 있기 때문이다. 서비스 제거 연산은 시간이 갈수록 연산 시간이 점점 감소한다. 그림 19에서 로컬 서비스 등록, 원격 서비스 등록, 서비스 수정과 삭제의 평균 연산 시간은 각각 59.0646ms, 40.2767ms, 114.2569ms, 52.14263ms 이다. 단일 서비스 검색 연산은 해싱 기반의 검색으로 오직 0.0012ms을 가진다. 전체 서비스 나열 연산은 서비스의 수에 비례한다. 10개의 서비스를 등록한 상태에서 전체 서비스 나열 연산은 평균 0.03212ms의 시간이 소요되었다.

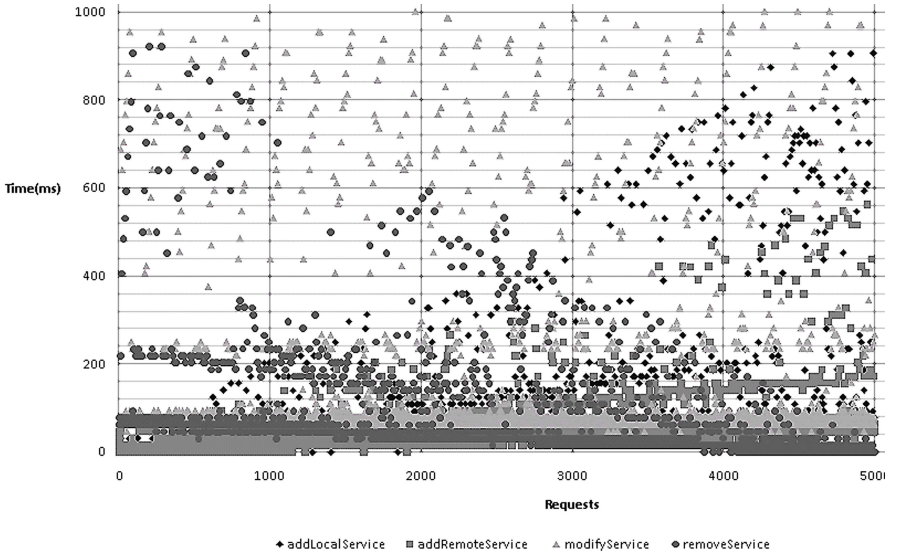

그림 19. DOKDO-WS의 서비스 레지스트리 연산

2) 도메인 객체 측면

(1) 정성적 평가

표 3과 표 4에서는 기존 도메인 객체 관련 연구와 DOKDO-WS를 비교하고 있다. 대부분의 연구에서 연결과 퍼시스턴스에 대한 자체적인 API를 제공한다. 데이터 저

장 방식은 파일 시스템, XML DB, RDBMS로 나뉜다. 그러나 동시성 문제나 트랜잭션 처리, 보안 등이 기본적으로 요구되는 웹 서비스 환경에서는 파일 시스템을 퍼시스턴스 계층으로 사용하는 것은 현실적이지 않다. 매핑 관계 항목에서는 퍼시스턴스 계층으로 부터 객체를 복원하고 퍼시스턴스 계층으로 저장하는 관계를 표현한다. 도메인 객체 생성 방식은 대부분 수동 방식이며 자동 방식은 생산성을 높여준다. 학습 주기는 사용하는 도구의 수, 사용된 개념의 수, 사용 모듈의 수를 기준으로 얼마나 사용하기 쉬운지를 나타낸다. 예를 들어, Low는 경량의 접근법을 제공하는 것을 의미한다.

표 3. 도메인 객체 관련 연구의 특징 비교 - I

항목	Taming XML	Naked Object	Runtime Generation	Decoupling OO Layer	Universal Relation Data Source
연결방식	API	API	External ORM	API	API
퍼시스턴스 API	×	×	External ORM	○	○
퍼시스턴스 타입	File	File	RDBMS	RDBMS	RDBMS
매핑 관계	Object<->XML or XIR	Properties->Object<->GUI	Object->Dynamic Proxy<->Persistence	DBMS<->Object	DBMS<->Object<->Formatted String
도메인 객체 생성 방식	Manual(Marshalling)	Automatic	Depend on ORM	Manual	Manual
학습 주기	Low	Low	Middle	Low	Low

표 4. 도메인 객체 관련 연구의 특징 비교 - II

항목	JOXM	iBatis	Hibernate	DOKDO-WS
연결방식	API	API	API	API
퍼시스턴스 API	○	○	○	○
퍼시스턴스 타입	XML DBMS	RDBMS	RDBMS	RDBMS
매핑 관계	XMLDBMS<->Object->XML	DBMS<->SQLMap<->Object	DBMS<->E-RMap<->Object	DBMS<->Model Map<->Object
도메인 객체 생성 방식	Manual(XPath)	Manual	Manual	Semi-Automatic
학습 주기	Low	Low	High	Low

(2) 정량적 평가

① 삽입, 수정, 삭제, 복합 연산

그림 20은 MS-SQL DBMS와 JOXM에서의 삽입, 수정, 삭제와 복합 연산에 대한 성능 평가를 보인다. registerCustomer 연산은 9개의 열을 가진 행을 입력하며, registerAccount 연산은 5개의 열을 가진 행을 입력한다. JOXM은 입력 행의 크기에 크게 영향을 받는 것을 알 수 있다. 수정 연산인 modifyCustomer 연산은 입력 연산보다 JOXM은 4배, 나머지는 6배 정도 성능이 낮았다. 삭제 연산인 unregisterCustomer 연산은 입력 연산 보다 4배 정도 낮았다. 복합 연산인 deposit과 withdrawal 연산에서는 각각 덧셈, 뺄셈 연산을 더하고 값의 변화량을 다른 테이블에 삽입 연산한다. 이 때 추가한 연산의 종류에 상관없이 복합 연산의 성능은 입력 연산 보다 JOXM에서 3.5배 정도 낮고, 나머지에서는 11배 정도 낮다. 결과적으로 MS-SQL DBMS에서 제안한 프레임워크에서 사용하는 DOI 프레임워크가 가장 높은 성능을 가지며, 차례로 iBatis, Hibernate, JOXM 순이다.

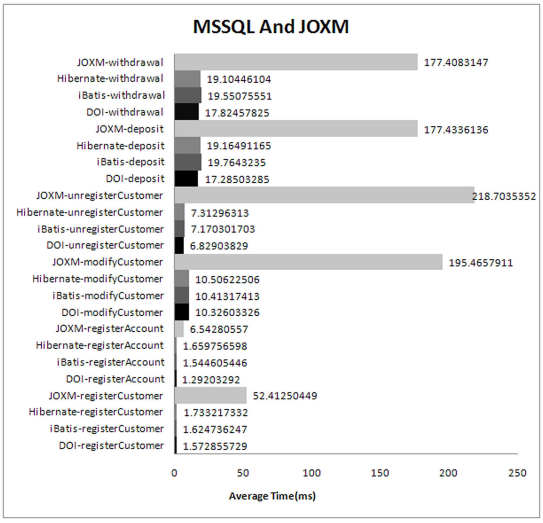

그림 20. MS-SQL과 JOXM에서의 삽입, 수정, 삭제 연산

그림 21에서 삽입, 수정, 삭제, 복합 연산을 MySQL DBMS에서 수행한 성능 평가를 보인다. 삽입 및 삭제 연산에서는 DOI, 수정 연산에서는 Hibernate, 복합 연산에서는 iBatis가 가장 높은 성능을 보인다. DOI는 복합 연산에서 가장 느린 성능을 보였지만, 나머지 연산에서는 전반적으로 높은 성능을 보인다.

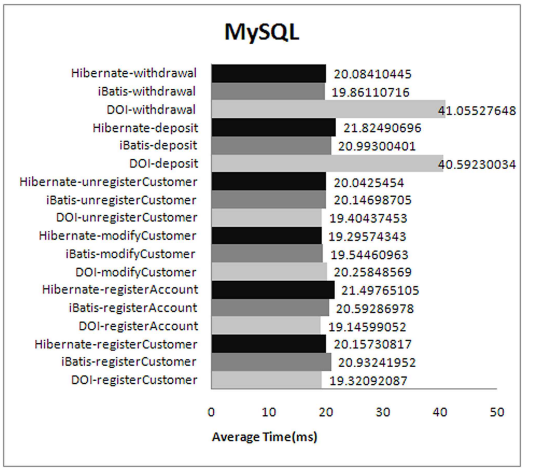

그림 21. MySQL에서의 삽입, 수정, 삭제 연산

그림 22에서 삽입, 수정, 삭제, 복합 연산을 Oracle DBMS에서 수행한 성능 평가를 보인다. 삽입, 수정, 삭제 연산에서 iBatis가 가장 높은 성능을 보인다. 그러나 복합 연산에서는 DOI가 iBatis와 Hibernate 보다 성능이 높다. 특히 withdrawal 연산에서 Hibernate보다 3배, iBatis보다 1.3배 정도 성능이 높다.

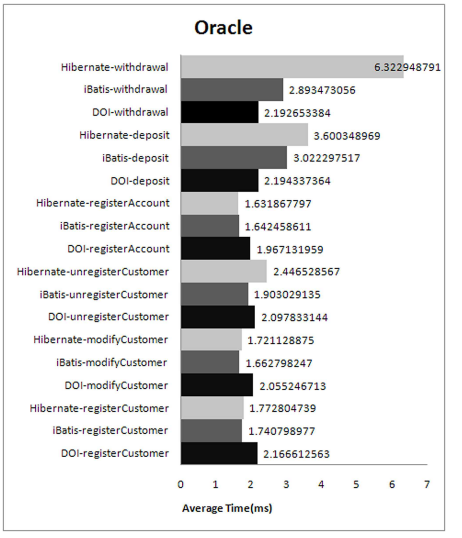

그림 22. Oracle에서의 삽입, 수정, 삭제 연산

② 선택 연산(포인트 질의, 범위 질의)

선택 연산은 9개의 열을 가진 Customer 테이블에 대해서 100 행을 반환하는 범위 질의와 특정 id 값을 가진 행을 반환하는 포인트 질의로 나누어서 테스트한다. 그림

23은 MS-SQL에서의 선택 연산에 대한 테스트를 보인다. ‘client’로 끝나는 연산은 사용자 수신 시간이다. 포인트 질의에서 DOI는 iBatis나 Hibernate에 비해서 52배 정도 높은 성능을 보인다. 네트워크 소요 시간은 대부분의 경우 모든 프레임워크가 15ms 정도로 비슷했으나 범위 질의에서 iBatis는 2배 정도 느린 29ms가 소요된다. 범위 질의에서 DOI가 iBatis보다 1.2배, Hibernate보다 2배 정도 높은 성능을 보인다. 결국 MS-SQL에서 선택 연산은 DOI가 가장 높은 성능을 보인다.

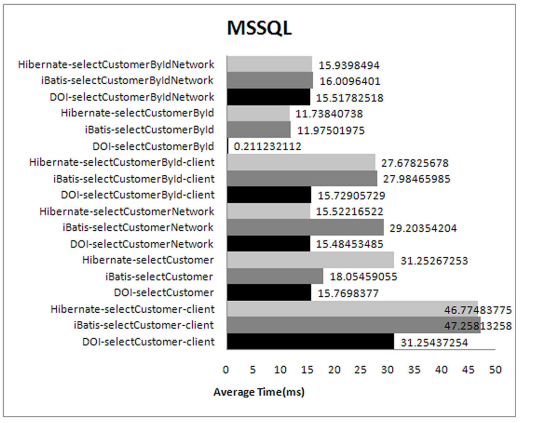

그림 23. MS-SQL에서의 선택 연산

그림 24는 MySQL에서의 선택 연산에 대한 성능 평가를 보인다. JDBC 드라이버의 특성 탓으로 DOI는 다른 프레임워크에 비해 가장 나쁜 성능을 보였다. 다른 프레임워크의 경우 메모리상에서 결과 집합을 반환하나 DOI의 경우엔 물리 디스크로부터 결과 집합을 반환했기 때문이다. 그러나 이 연산을 C 언어 환경에서 평가를 할 경우 DOI가 다른 ORM에 비해서 5배 정도 높은 성능을 보인다.

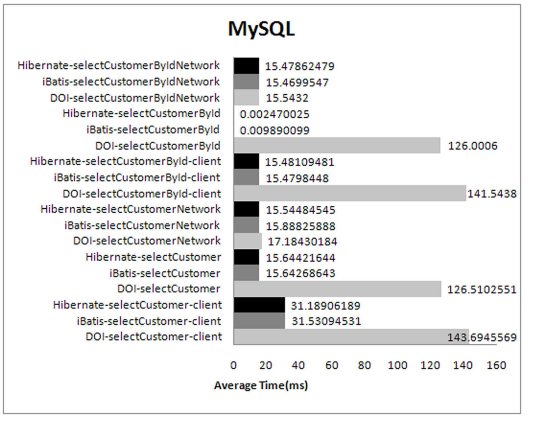

그림 24. MySQL에서의 선택 연산

그림 25는 Oracle에서의 선택 연산에 대한 성능 평가를 보인다. 포인트 연산에서는 Hibernate와 iBatis가 훨씬 높은 성능을 보이지만 0.019ms 내외의 작은 차이이다. 반면 범위 질의에서 DOI가 82배 정도 높은 성능을 보이며, 시간도 15ms 정도 차이가 난다. 그 결과, Oracle에서 선택 연산은 DOI가 다른 프레임워크 보다 성능이 높은 것을 알 수 있다.

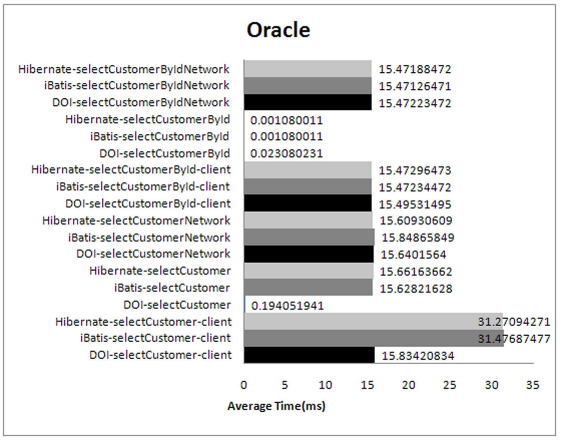

그림 25. Oracle에서의 선택 연산

3. 구현

1) 경량 서비스 저장소

DOKDO-WS 프레임워크는 [2]의 연구에서 언급했던 대부분의 서비스 저장소의 요건을 만족한다. 바이너리의 크기는 128Kbytes 이며, 빠른 초기화 속도를 가진다. API로 제공되는 컴포넌트들은 변경 없이 사용하기 편하며, 기존의 인프라 구조의 서비스들도 변경할 필요가 없이 재사용할 수 있다.

서비스 제공자는 서비스에 대한 등록, 수정, 삭제는 일반적인 웹 브라우저를 통해서 수행할 수 있고, 서비스 소비자는 즉시 변경된 내용을 확인할 수 있다. 다른 서비스 레지스트리를 하나의 서비스로써 제공할 수 있기 때문에 다른 웹 서비스 레지스트리와의 통합도 쉽다. 게다가 서비스 제공자가 직접적으로 웹 서비스 레지스트리를 제공하기 때문에 부적절한 서비스가 제공되지 않는다. 로컬 서비스는 현재 도메인의 특정 서비스에 종속되며 서비스 제공자는 원격의 서비스도 동일 도메인에서 제공할 수 있다.

2) 도메인 객체 매핑

단순한 메타 모델을 통해서 도메인 객체와 DBMS, XML 간을 연결한다. 기본적인 데이터 타입을 정의하며 데이터 객체의 직렬화 혹은 역직렬화와 서비스 인스턴스의 마샬링 혹은 언마샬링을 수행한다. 도메인 객체의 타입 변환은 변환기 서비스를 통해서 이뤄진다. 또한 퍼시스턴스 계층에 쉽게 접근하기 위한 연산은 자동으로 생성한다. 복잡한 퍼시스턴스 연산은 서비스 제공자가 수동으로 추가할 수 있다.

3) 동적 조합

서비스는 서비스 실행 계층, 서비스 뷰 계층을 통해서 동적으로 조합될 수 있다. 그림 26은 서비스 실행 계층에서의 조합 알고리즘을 보인다. 서비스 실행 계층에서 실행할 연산 전 후에 실행 코드를 추가한 동적 프록시를 서비스 인스턴스로 생성하여 제공하게 된다. 동적 프록시는 Reflection을 이용하여 서비스를 주입하거나 AOP를 통해 바이트 코드를 변경함으로써 생성된다.

```
Let S = {B, A}, D = {s0, s1, s2, ... sn} where S is a Service Prototype, s is
Service Instance and D is Dynamic Proxy which is mixed of s. HI and TI
means Host group id and target group id.
W[GI] is the set of Services stored in WSDLRepository.
[:] is instantiation operator
[T] is type operator
[←] is assignment
Input : [Host Group Id] and [Target Group Id(TI), Position]
Output : Dynamic Proxies
01 foreach Sh in W[HI]
02    D[HI] ← :Sh
03    foreach St in W[TI]
04       if TSh = B then D[HI].B ∪ St
05       else D[HI].A ∪ St
06       end if
07    end foreach
08 end foreach
09 return D
```

그림 26. 서비스 실행 계층에서의 조합 알고리즘

그림 27은 서비스 뷰 계층에서의 조합 알고리즘을 보인다. 서비스 뷰 계층에서는 함수형 언어를 기반으로 함수들을 조합하여 서비스 그룹을 생성한다. 서비스 모델 계층의 서비스 프로토타입에는 성공과 실패에 대한 함수 그룹에 대한 선언이 없다. 그렇지만 서비스 뷰 계층에서는 함수를 제 1 클래스 객체로 사용하기 때문에 동적으로 함수 그룹을 생성할 수 있다. 성공 실패 함수를 가진 함수 그룹은 조합된 서비스 인스턴스로 동작하게 된다. 이외에도 조합에 대한 설정을 런타임에 접근하여 함수 그룹을 생성할 수도 있다.

```
Let E = {S, F}, E, F = {f0, f1, f2, ... fn} where E is a Service Execution
Function Group, S is set of succeed Functional Factor and F has opposite
meaning.
W[GI] is the set of Services stored in WSDLRepository.
Input : [Host Group Id] and [Target Group Id(TI), Position]
Output : Execution Function Group
01 foreach W[HI] Sh
02    foreach W[TI] St, E
03       E.S ∪ St.S
04       E.F ∪ St.F
05       end if
06    end foreach
07 end foreach
08 return E
```

그림 27. 서비스 뷰 계층에서의 조합 알고리즘

서비스 실행 계층에서 생성되는 동적 프록시는 성능이 높고, 서비스 뷰 계층에서 접근할 때 단일 서비스로 인식된다. 게다가 오류 검출도 쉽다. 그러나 AOP 개념으로 바이트 코드를 직접 수정하는 방식은 매우 복잡하기 때문에 Reflection을 이용하여 동적 프록시를 생성하는 방법을 사용하는 것이 더 좋다. 반면, 서비스 뷰 계층에서 조합하는 것은 성능이 더 낮고 오류 검출이 어렵지만 그것은 편리성과 확장성을 제공한다.

4) 유연한 실행

Reflection을 통해 메타 데이터를 획득하고 WSDL을 생성하기 때문에 기존에 존재하는 컴포넌트를 웹 서비스로 쉽게 공개할 수 있다. 이 과정에서 별도의 추가적인 도

구가 필요하지 않고, 기존에 존재하는 인프라 구조를 변경 없이 재사용할 수 있다. 서비스 실행 과정에서 컨텍스트 유지를 위해서 세션을 지원하며 동일한 서비스에 동일한 이름의 연산에 다른 파라미터를 전달하여 호출할 수 있도록 해주는 오버로드를 제공한다. 또한 원격과 지역에 존재하는 서비스를 모두 호출할 수 있다. 또한 로컬 서비스 종점 URL은 자동으로 생성되기 때문에 서비스 제공자는 로컬 서비스 종점에 대해서 신경 쓸 필요가 없다.

5) 웹 서비스 탐색기

그림 28은 기본적으로 제공되는 웹 서비스 탐색기를 보인다. 탐색기는 메타 서비스 그룹의 DOIMetaService 서비스를 통해서 WSDL 저장소의 모든 서비스를 원격과 지역으로 구분하여 표시한다. 서비스 목록은 XML 형태로 반환되기 때문에 서비스 이용자에게 제공할 때는 뷰 계층에서 제공하는 XSLT 프로세서 서비스를 이용하여 원하는 형태의 사용자 인터페이스를 생성한다. 기본적으로 제공되는 템플릿을 통해서 웹 서비스 제공자는 외부로 배포할 서비스를 선택 및 실행해 볼 수 있다.

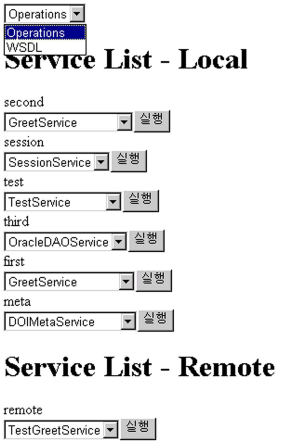

그림 28. 기본 웹 서비스 탐색기

그림 29는 특정 서비스에 있는 연산을 나열하는 인터페이스이다. 연산에 매개변수가 있다면 입력 폼을 제공하고 서비스 제공자가 서비스 연산을 실행할 수 있다. 서비스 인스턴스가 프레임워크 인지 클래스인 AbDOIService 클래스를 기반으로 생성된 서비스 인스턴스라면 getSessionId(), isWarmed() 메서드를 반드시 포함하게 된다. 이는 DBMS로의 연결과 요청, 응답, 세션에 관련된 객체를 사용할 수 있게 만든다.

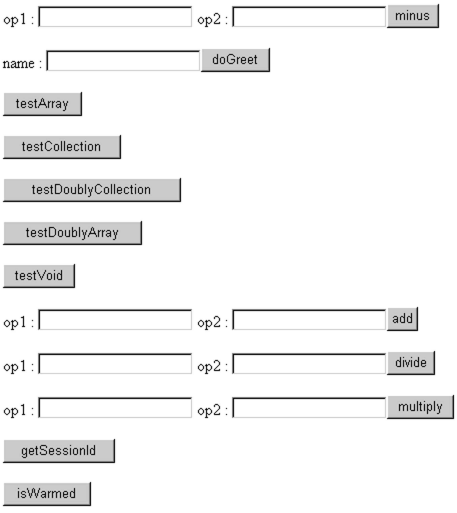

그림 29. 서비스 연산 나열

그림 30은 지역 서비스를 실행한 결과 XML 문서를 보여준다. 지역 서비스는 직접 서비스 인스턴스에 접근하여 실행한다.

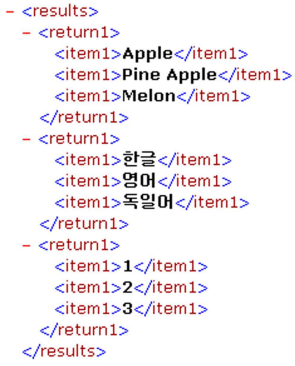

그림 30. 웹 서비스 실행 - 지역

원격 서비스의 경우 그림 29와 동일한 폼을 생성하지만 실행할 때의 과정은 다르다. 그림 31은 원격 서비스의 실행 결과로 반환된 SOAP 메시지를 보인다. 원격 서비스는 WSDL 문서에서 연산을 추출하여 컨텍스트를 구성하고 SOAP 요청을 보낸다.

그림 31. 웹 서비스 실행 - 원격

그림 32는 이기종 환경인 .NET 프레임워크에서 WSDL 문서를 참조하여 웹 서비스 프록시를 생성하는 것을 보인다. 그림 33은 자바로 구현된 DOKDO-WS 프레임워크에 SOAP 메시지로 요청하여 처리된 결과를 보인다.

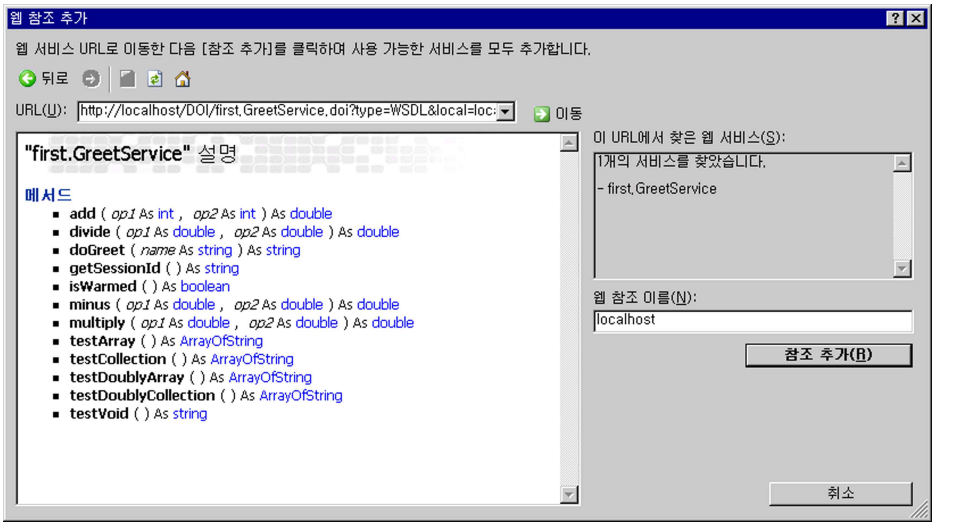

그림 32. .NET 웹 서비스 참조 추가

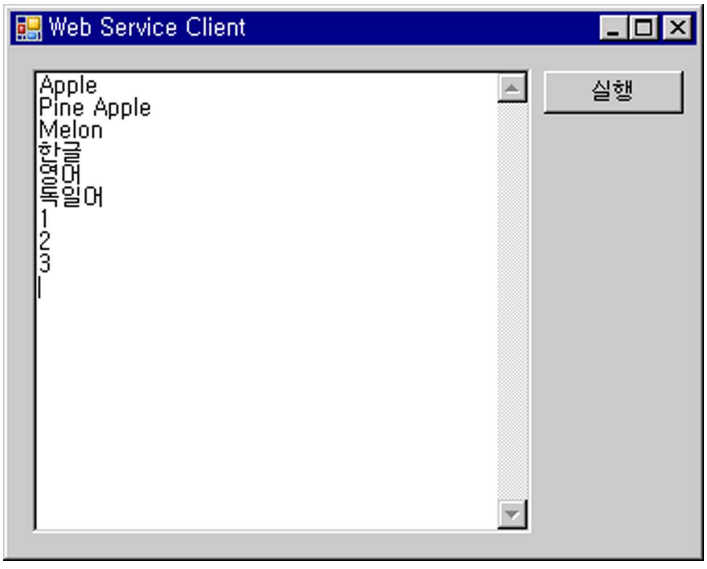

그림 33. .NET 웹 서비스 실행

V. 결론 및 향후 연구

논문에서는 기존 응용 프로그램 개발 방법론에 큰 변화를 가하지 않고 웹 서비스 환경에 적용할 수 있는 DOKDO-WS 프레임워크를 제안한다. 웹 서비스 제공자는 이용이 편리하고 확장성이 높은 프레임워크를 필요로 한다. 그러나 기존의 웹 서비스 프레임워크는 효율성과 현실성을 모두 만족시키지 못했고 경량의 접근법이 아니었다. 프레임워크는 서비스를 검색 및 실행 할 수 있는 기능을 제공하고 실행 시간에 서비스 변경을 사용자가 즉시 인지할 수 있도록 해야 한다. 서비스 소비자는 HTTP 기반의 레거시 웹 환경과 새로운 웹 서비스 기술이 단절되지 않고 기존의 HTTP 기반의 요청 응답 방식도 유지할 수 있길 원한다. 그래서 DOKDO-WS 프레임워크는 SOAP과 기존의 HTTP 요청, 응답, 그리고 실행 사이에서 상태를 유지하기 위한 세션을 모두 지원한다.

DOKDO-WS 프레임워크는 데이터 모델링과 반복되는 입력, 검색, 수정, 삭제의 퍼시스턴스 연산을 자동화 해준다. 웹 서비스 기반의 응용 프로그램에서 사용되는 비즈니스 로직은 언어나 환경에 따라 크게 다를 수 있다. 그러나 거기에서 사용되는 데이터 모델은 크게 변경되지 않는다. ER 모델은 여전히 계속 이용되고 있고 시멘틱 데이터 모델의 저장과 복원을 모두 XML로 할 수는 없다. 성능 평가에서 XML 기반의 접근법은 데이터 저장과 복원에 최적화된 ER 모델에 비해 많이 떨어진 성능을 보였다. 본 논문에서는 ER 모델에서 성능을 얻고 XML 기반의 언어에서 확장성을 얻음으로써 장점을 취했다. 즉, XML 스키마 기반의 서비스, 데이터, 그리고 뷰 모델을 제안했다. 비록 본 논문에서 사용한 도메인 모델이 RDF와 OWL과 같은 표준 도메인 모델이 아니지만 DOI는 모든 XML 기반의 도메인 모델을 다룰 수 있기 때문에 서비스 제공자는 이를 이용해서 XML 기반의 어떠한 도메인 모델이라도 다룰 수 있을 것이다.

향후 연구는 표준 세션 지원, 캐시 문제 및 시멘틱 웹을 위한 추론 엔진에 대한 것이다. 현재 세션 구현은 특정 환경에서만 사용 가능한 부분적인 설계 및 구현을 제공한다. 그러므로 미래에 표준 WS-Session이 구현될 것이다. 성능 평가에서 보이듯이 다른 프레임워크와 견주어서 더 높은 성능을 보인다. 캐시를 이용하면 현재보다 더 높은 성능을 보일 수 있을 것이다. 추론 엔진은 구현된 동적인 조합 엔진에 추가되어 시멘틱 웹을 구현하는데 이용될 것이다.

참고문헌

[1] Okkyung Choi, Jungwoo Lee, Sangyoung Han, Advanced Web Services Retrieval System using Matchmaking Algorithm. Journal of the Korea Intelligent Information Systems Society, Volume 13. No 3 pp. 1~15. 2007

[2] Martin Treiber, Schahram Dustdar, Active Web Service Registries, IEEE Internet Computing, Volume 11, Issue 5, pp. 66 - 71, 2007

[3] Jianxun Liu, Jie Liu, Lian Chao, Design and Implementation of an Extended UDDI Registration Center for Web Service Graph, IEEE International Conference on Web Services, pp. 1174 - 1175, 2007

[4] Lin Liang, Wenge Rong, Kecheng Liu, Intelligent Agents for Pragmatic Web Services, Sixth International Conference on Advanced Language Processing and Web Information Technology, pp. 530 - 536, 2007

[5] N.W. Lo, Chia-Hao Wang, Web services QoS evaluation and service selection framework - a proxy-oriented approach, TENCON 2007 - 2007 IEEE Region 10 Conference, pp. 1 - 5, 2007

[6] Eyhab Al-Masri, Qusay H. Mahmoud, WSCE: A Crawler Engine for Large-Scale Discovery of Web Services, IEEE International Conference on Web Services, pp. 1104 - 1111, 2007

[7] Jeong-Youn Yu, Kyu-Chul Lee, Ontology describing Process Information for Web Services Discovery, The Journal of Society for e-Business Studies, Volume 12. No 3, pp. 151 - 175, 2007

[8] Hyun Namgoong, Moonyoung Chung, Kyung-il Kim, HyeonSung Cho, Yunku Chung, Effective Semantic Web Services Discovery using Usability, Advanced Communication Technology The 8th International Conference Volume 3, pp. 2199 - 2203, 2006.

[9] Assia Ben Shil, Mohamed Ben Ahmed, Additional Functionalities to SOAP, WSDL and UDDI for a Better Web Services' Administration, Information and Communication Technologies, ICTTA '06. 2nd. Vol. 1, pp. 572 - 577, 2006

[10] S.M.F.D Syed Mustapha, Relation-Based Case Retrieval Approach for Web Services Selection. IEEE/WIC/ACM International Conference on Web Intelligence. pp. 644 - 648. 2006

[11] Won-Suk Lee, Jong-Hun Park, Kyu-Chul Lee. The Protocol on WS-ECA Framework, Journal of Korean Society for Internet Information, Vol. 8. No 6. pp. 55 - 73. 2007

[12] Won-Suk Lee, Dong-Min Shin, Kyu-Chul Lee. Design and Implementation of WS-ECA Framework, The Journal of the Korean Institute of Maritime Information and Communication

Sciences, Volume 11. No 8. pp.1612-1618. 2007

[13] Wu Chou, Li Li, Feng Liu. Web Services Methods for Communication over IP, IEEE International Conference on Web Services, pp. 372 - 379, 2007

[14] Yanlong Zhai, Hongyi Su, Shouyi Zhan, A Reflective Framework to Improve the Adaptability of BPEL-based Web Service Composition, IEEE International Conference on Services Computing, Vol. 1, pp. 343 - 350, 2008

[15] Freddy Lecue, Alexandre Delteil, Alain Leger, Applying Abduction in Semantic Web Service Composition, IEEE International Conference on Web Services, pp. 94 - 101, 2007

[16] Karthikeyan Ponnalagu, N.C. Narendra, Jayatheerthan Krishnamurthy, R. Ramkumar, Aspect-oriented Approach for Non-functional Adaptation of Composite Web Services, IEEE Congress on Services, pp. 284 - 291, 2007

[17] Hui Kang, Xiuli Yang, Sinmiao Yuan, Modeling and Verification of Web Services Composition based on CPN, NPC Workshops. IFIP International Conference on Network and Parallel Computing Workshops, pp. 613 - 617, 2007

[18] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, Cheng-Hung Chen, On Composing a Reliable Composite Web Service: A Study of Dynamic Web Service Selection, IEEE International Conference on Web Services, pp. 184 - 191, 2007

[19] Wen-Yau Liang, Apply Rough Set Theory into the Web Services Composition, 22nd International Conference on Advanced Information Networking and Applications, pp. 888 - 895, 2008

[20] Il-Woong Kim, Kyong-Ho Lee, A Model-Driven Approach for Converting UML Model to OWL-S Ontology, Journal of KISS:Computing Practices and Letters, Vol. 13, No 3, pp. 179 - 192. 2007

[21] Yongyan Zheng, Paul Krause. Asynchronous Semantics and Anti-patterns for Interacting Web Services, Sixth International Conference on Quality Software, pp. 74 - 84, 2006

[22] W.L Yeung, Mapping WS-CDL and BPEL into CSP for Behavioral Specification and Verification of Web Services, 4th European Conference on Web Services, pp. 297 - 305, 2006

[23] Juil Kim, Woojin Lee, Kiwon Chong, A Tool to Construct a Web Service by Synthesizing Several Web Services, International Conference on Hybrid Information Technology, Vol 2, pp. 530 - 535, 2006

[24] Feng Liu, Gesan Wang, Li Li, Wu Chou, Web Service for Distributed Communication Systems, IEEE International Conference on Service Operations and Logistics, and Informatics, pp. 1030 - 1035, 2006

[25] Wu Chou, Li Li, Feng Liu. Web Services for Service-Oriented Communication, International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 1 - 8,

2006

[26] Ricardo Lemos Vianna, Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco, Lisandro Zambenedetti Granville, Investigating Web Services Composition Applied to Network Management, International Conference on Web Services, pp. 531 - 540, 2006

[27] Sam Chung, Jennifer R. Pan, Sergio Davalos, A Special Web Service Mechanism : Asynchronous .NET Web Services, Telecommunications, International Conference on Internet and Web Applications and Services/Advanced. pp. 212 - 212. 2006

[28] Costas Vassilakis, George Lepouras, Akrivi Katifori. Web Service Execution Streamlining, International Conference on Service Systems and Service Management, Vol. 2, pp. 1564-1569. 2006

[29] Federica Paci, Mourad Ouzzani, Massimo Mecella. Verification of Access Control Requirements in Web Services Choreography, IEEE International Conference on Services Computing, Vol. 1.pp. 5 - 12, 2008.

[30] Hyun Sik Hwang, Hyuk Jin Ko, Kyu Il Kim, Ung Mo Kim. Agent-Based Delegation Model for the Secure Web Service in Ubiquitous Computing Environments, International Conference on Hybrid Information Technology, Vol. 1. pp. 51 - 57, 2006

[31] Pat. P.W. Chan, Michael R. Lyu. Dynamic Web Service Composition: A New Approach in Building Reliable Web Service, 22nd International Conference on Advanced Information Networking and Applications, pp. 20 - 25, 2008

[32] Lei Deng, Jian Wu, Zhengguo Hu. ISCF: A Semantic Web Service Composition Framework Based on OAA, The 3rd International Conference on Grid and Pervasive Computing Workshops, pp. 250 - 255, 2008

[33] Ning Gu, Juntao Cui, Wei Ye, Haixun Wang, Jian Pei. A System Framework for Web Service Semantic and Automatic Orchestration, 2nd International Conference on Pervasive Computing and Applications, pp. 606 - 611, 2007

[34] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, Proceedings of the Second International Semantic Conference, Vol 2870 of LNCS. pp. 227 - 247. 2003.

[35] Rohit Aggarwal, Kunal Verma, John Miller, William Milnor. Dynamic Web Service Composition in METEOR-S, LSDIS Lab Technical Report. 2004

[36] Evren Sirin, Bijan Parsia, James Hendler. Template-based Composition of Semantic Web Services. AAAI Fall Symposium on Agents and the Semantic Web. 2005

[37] John Domingue, Liliana Cabral, Farshad Hakimpour, Denilson Sell, Enrico Motta. IRS-III: A

Platform and Infrastructure for Creating WSMO-based Semantic Web Services, Proceedings of the Workshop on WSMO Implementations. 2004

[38] Denilson Sell, Farshad Hakimpour, John Domingue, Enrico Motta, Roberto C.S. Pacheco. Interactive Composition of WSMO-based Semantic Web Services in IRS-III. Proceedings of the First AKT Workshop on Semantic Web Services. 2004

[39] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, Dana Nau. HTN Planning for Web Service Composition Using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 1, No. 4. pp. 377-396. 2004

[40] Shankar R. Ponnekanti, Armando Fox. SWORD: A Developer Toolkit for Web Service Composition, Proceedings of the 11th International World Wide Web Conference, 2002

[41] Harald Meyer, Hagen Overdick, Mathias Weske. Plaengine: A System for Automated Service Composition and Process Enactment, Proceedings of the WWW Service Composition with Semantic Web Services, p. 3-12. 2005

[42] Muhammad Ahtishame Aslam, Jun Shen, Sőren Auer, Michael Herrmann. An Integration Life Cycle for Semantic Web Services Composition. 11th International Conference on Computer Supported Cooperative Work in Design, pp. 490 - 495, 2007

[43] Jin-han Kim, Chang-ho Lee, Jae-Jeong Lee, Byung-Jeong Lee. A Framework For Web Service Evolution using UML and OWL-S. Journal of Digital Contents Society, Vol. 8. No 3. pp.269-277. 2007

[44] Matt Bone, Peter F. Nabicht, Konstantin Läufer and George K. Thiruvathukal. Taming XML: Objects First, Then Markup, IEEE International Conference on Electro/Information Technology, pp. 488 - 493, 2008

[45] Konstantin Läufer. A Stroll through Domain-Driven Development with Naked Objects, Computing in Science & Engineering Vol 10. Issue 3. pp. 76 - 83, 2008

[46] Angela Nicoara, Gustavo Alonso. Making Applications Persistent at Run-time, IEEE 23rd International Conference on Data Engineering, pp. 1368 - 1372, 2007

[47] Fabio Dias Fagundez, Ricardo Niederberger Cabral, Gustavo Melim do Carmo. Decoupling Relational Database from Object Oriented Layers, Fourth International Conference on Information Technology, pp. 870 - 871. 2007

[48] Vit Vrba, Lubomir Cvrk, Vit Novotny, Karol Molnar. Architecture of a universal relation data source for web applications with advanced access control and simplified migration, Proceedings of the International Conference on Software Engineering Advances. pp. 68 - 68. 2006

[49] Adam Dukovich, Jimmy Hua, Jong Seo Lee, Michael Huffman, Alex Dekhtyar. JOXM: Java Object

- XML Mapping, 8th International Conference on Web Engineering. pp. 332 - 335, 2008

[50] Andrea D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. International Conference on Web Services, pp. 789 - 796, 2006

[51] William R. Cook, Janel Barfield. Web Services versus Distributed Objects : A Case Study of Performance and Interface Design, International Conference on Web Services. pp. 419 - 426, 2006

Abstract

A Framework based on Domain Object Interface for Web Services in heterogeneous Distributed Environments

Lim, Eun-Cheon
Department of Multimedia Engineering
The Graduate School
Sunchon National University
(Supervised by Prof. Sim, Choun Bo)

An object oriented software defines many objects and is operated by communication between objects. Object oriented softwares in distributed environment are constructed by web service. It requires searchablity, effectiveness, reliability, stability, and availability for services and accessibility to domain objects and execution performance to implement successfully web services in distributed environment.

In practice, it is hard to realize all of the fundamental architecture to build a web service application so that researches for the development of web services with frameworks has kept going on. Researches for web service framework resolved overall problems that are occurred on registration and search for the web service, communication protocol, orchestration, and execution, but developer should consider much about standard web service technic if he or she manipulate business logic with web services. The problem is occurred when displaying data sources to users in legacy web application after making a connection between a web service framework and existing data sources, namely, the problems are mapping between domain

objects and data sources, connection between local business logic, configuration and deployment of the business logic for web service, and complexity of the execution. It requires developers additional complexity because they cannot use existing development model so that service provider need a lightweight framework that minimizes wide fluctuation of existing development model.

In this paper, we propose a *Meta model* of the web service for devising convenience on searching, configuration, execution and a *Domain Object Interface(DOI)* to handle *Meta model* and design and implement a *Distribution Oriented Knowledge-base of Dynamically Operable Web Servcies(DOKDO-WS)* framework based on the *DOI*. The proposed DOKDO-WS framework offer a web service registry to enroll and search business logics in a domain and a web service browser to identify and execute and a functionality to orchestrate dynamically or statically. Since service is added if that pass the verification process so only executable services should be shown in runtime. Local services is executed by reflection and remote services is executed by the SOAP proxy instances and services that navigated by web service browser is executed by popular web browsers. At the same time, the meta model of the proposed DOKDO-WS framework is defined by XML Schema and is divided into data meta model and service meta model. The service meta model defines local and remote services. The data meta model is used to generate automatically tables, views and stored procedures that have shown high performance in the most persistence operations. Domain objects are transferred in form of the XML by XML transformers and are viewed to user on an appropriate GUI by XSLT processing.

We evaluate performance of the service registry at the factor of processing time to verify the effectiveness, stability, accessibility of the framework and carry out performance evaluation about the transaction processing in a bank application to evaluate processing power of the DOI. In service registry, average times on enrolling local services, enrolling remote services,

modifying services, and removing services takes times on average each 59.0646ms, 40.2767ms, 114.2569ms, and 52.14263ms. Operations on the single service are based on hashing and takes only 0.0012ms on the average. The operation of the enumeration on the total services are increased or decreased in proportion to the number of services, it takes 0.03212ms when there are 10 services registered. In performance evaluation on processing of the domain objects, we performed tests on the factors that are DBMS, ORM framework, the type of persistence operations. In the MS-SQL, Oracle DBMS, the proposed DOI surpasses other frameworks in insert, modify, delete, composite, and select operation, especially at select operation that is frequently used in web service environment the DOI wins an overwhelming result. However, in the MySQL DBMS, get a sluggish performance, which caused by faults on tested JDBC driver. The DOI is faster 5 times than other ORM frameworks in evaluation on the C language. The DOI make 0~3 errors on 100,000 requests. Based on performance evaluation, the DOKDO-WS verify that has higher effectiveness, stability, and reliability as a service registry and an execution engine.

Key Words : web service framework, web service registry, web service orchestration, web service execution engine, domain object, ORM, SOA