韓國컴퓨터情報學會 論文誌
第14卷 第9號, 2009. 9.

2009-14-9-1-1

관점지향 프로그래밍 및 리플렉션 기반의 동적 웹 서비스 조합 및 실행 기법

임 은 천*, 심 춘 보**

A Dynamic Web Service Orchestration and Invocation Scheme based on Aspect-Oriented Programming and Reflection

Eun-Cheon Lim*, Chun-Bo Sim**

요 약

웹 서비스 조합 분야는 단일 서비스를 재사용하여 가치 있는 서비스를 생성하기 위해 등장했으며, 최근에는 차세대 웹 서비스인 시멘틱 웹을 구현하기 위해 IOPE를 기반으로 단순 검색 및 조합 대신에 규칙이나 AI를 통한 검색 및 조합 방법이 제안되고 있다. 또한 보다 효율적인 모듈화를 위해 기존의 객체지향 프로그래밍 방식보다는 관점지향 프로그래밍 방식이 도입되고 있다. 본 논문에서는 시멘틱 웹을 위해 관점지향 프로그래밍(Aspect-Oriented Programming, AOP) 및 리플렉션(Reflection)을 적용한 동적 웹 서비스 조합 및 실행 기법을 설계한다. 제안하는 기법은 웹 서비스의 메타 데이터를 동적으로 획득하기 위해 리플렉션 기법을 사용하고 아울러 동적으로 웹 서비스를 조합하기 위해 AOP 기반 접근방식을 통해 바이트 코드를 생성한다. 또한 리플렉션을 이용한 동적 프록시 객체를 통해 조합된 웹 서비스를 실행하는 방식을 제안한다. 제안하는 기법의 성능 평가를 위해 비즈니스 로직 계층과 사용자 뷰 계층 측면에서 조합된 웹 서비스를 검색하는 것에 대한 실험을 수행한다.

Abstract

The field of the web service orchestration introduced to generate a valuable service by reusing single services. Recently, it suggests rule-based searching and composition by the AI(Artificial Intelligence) instead of simple searching or orchestration based on the IOPE(Input, Output, Precondition, Effect) to implement the Semantic web as the web service of the next generation. It

• 제1저자 : 임은천 교신저자 : 심춘보
• 투고일 : 2009. 08. 31, 심사일 : 2009. 09. 10, 게재확정일 : 2009. 09. 14.
* 순천대학교 멀티미디어공학과 석사졸업 ** 순천대학교 멀티미디어공학과 교수
※ 본 논문은 지식경제부 및 정보통신연구진흥원의 지원을 받아 수행된 연구결과(09-기반, 산업원천기술개발사업) 및 지식경제부의 대학IT연구센터지원사업의 연구결과로 수행되었음(IITA-2009-(C1090-0902-0047))

introduce a AOP programming paradigm from existing object-oriented programming paradigm for more efficient modularization of software. In this paper, we design a dynamic web service orchestration and invocation scheme applying Aspect-Oriented Programming (AOP) and Reflection for Semantic web. The proposed scheme makes use of the Reflection technique to gather dynamically meta data and generates byte code by AOP to compose dynamically web services. As well as, our scheme shows how to execute composed web services through dynamic proxy objects generated by the Reflection. For performance evaluation of the proposed scheme, we experiment on search performance of composed web services with respect to business logic layer and user view layer.

▸ Keyword : 웹 서비스 조합(Web Service Orchestration), 시멘틱 웹(Semantic Web), 관점지향 프로그래밍(Aspect-Oriented Programming), 리플렉션(Reflection)

Ⅰ. 서 론

웹 서비스 기술을 통해 기업의 비즈니스 로직을 처리하기 위한 다양한 연구가 활발히 진행 중에 있다. 기업은 기 구축된 비즈니스 워크플로우를 그대로 유지하면서 이기종의 분산 환경에서 다양한 서비스 확장 및 갱신을 지원하기 위해 동일한 인터페이스로 비즈니스 워크플로우를 처리하기를 원한다. 한편 최근 표준 웹 서비스 기술은 웹 서비스를 재사용하기 위한 언어로 WS-BPEL 및 WS-CDL 등이 제안되었으며, 특히 서비스 인스턴스의 실행 명세는 WSDL(Web Service Descripton Language) 언어로 배포되며 이기종의 환경에서 조합 언어에 명시된 형태로 이기종의 서비스 구현을 조합 언어에 바인딩하기 위한 연구가 진행 중이다[1]. 웹 서비스는 다양한 서비스를 조합하기 위해 비즈니스 프로세스와 동일한 도메인 내에서 조합에 참여하는 서비스를 검색하게 된다. 보다 정확한 서비스 검색을 위해 IOPE (Input, Output, Precondition, Effect)를 기반으로 한 검색에 시멘틱 모델링을 위한 언어가 도입되고 있다[2]. 그러나 웹 서비스 검색의 문제점은 서비스에 적합한 의미적으로 완벽한 온톨로지는 존재하지 않으며, 여러 온톨로지가 조합되어야 도메인에 최적의 온톨로지가 구축될 수 있다. 더우기 온톨로지는 계속 변화하며 서비스 개발 기간보다 온톨로지 구축 및 유지 보수 시간이 더 오래 소요된다. 조합을 하기 전에 동일한 의미를 가진 서비스라 하더라도 사용자마다 선호하는 서비스가 다를 수 있기 때문에 서비스의 위치, QoS, 기존 사용자들의 평판, 성능과 같은 다양한 정보를 사용자가 이를 확인하고 선택할 수 있도록 지원해야 한다[3].

관점지향 프로그래밍(AOP)은 기존의 객체지향 프로그래밍만으로는 완벽한 관심사의 분리가 어려운 문제를 보완하기 위한 프로그래밍 패러다임으로 여러 핵심 관심사에 걸쳐 나타나는 횡단 관심사를 애스펙트(Aspect)라는 모듈화 단위로 지원하는 프로그래밍 기법이다[4]. AOP는 별도로 정의된 관심사들이 필요에 의해 언제나 동적으로 플러그인(plug-in) 될 수 있게 해 주는 웨이빙(weaving) 메커니즘을 지원한다[5].

본 논문에서는 시멘틱 웹을 위해 관점지향 프로그래밍 및 리플렉션을 적용한 동적 웹 서비스 조합 및 실행 기법을 제안한다. 제안하는 기법은 서비스 인스턴스가 생성된 후에 이를 BCEL(Byte Code Engineering Library)과 같은 실행 코드 라이브러리를 이용하여 특정 서비스에 다른 서비스 인스턴스의 오퍼레이션을 주입하는 방식으로 동적 프록시를 생성한다. 또한 로컬 서비스 인스턴스의 메타 데이터 획득을 위해서 리플렉션 기법을 이용하며 원격 서비스 인스턴스의 메타 데이터 획득은 WSDL을 이용한다. 웹 서비스 검색을 하기 전에 사용자는 도메인에 존재하는 서비스를 확인하고 각 서비스 별 성능을 확인하여 서비스를 선택할 수 있다. 조합된 웹 서비스는 세션을 유지한 상태로 트랜잭션 및 보안을 처리할 수 있으며, 선언 시간이 아닌 실행 시간에 동적 프록시를 조합하고 실행할 수 있는 방법을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 연구의 배경 및 관련 연구를 살펴보고 3장에서는 제안하는 AOP 및 리플렉션 기반의 동적 웹 서비스 조합 및 실행 기법에 대해서 기술한다. 4장에서는 구현 결과 및 성능 평가를 소개하고 마지막으로 5장에서 결론 및 향후 연구를 제시한다.

II. 관련 연구

1. 웹 서비스 조합

웹 서비스가 검색된 후에 단일 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 있으며, 서비스 객체를 위한 메타 정보를 실행 시간에 조합하는 방법이다. 기존 연구에서 WS-BPEL, WS-CDL과 같은 표준 언어를 통해서 웹 서비스를 조합하고 실행하는 구조가 여기에 속하며[1][5] 특히 [1]의 연구에서는 웹 서비스를 조합할 수 있는 Reflection을 이용한 조합을 위해 프레임워크의 메타 모델과 BPEL 표준과의 매핑을 시도하였다. 아울러 [6]의 연구에서는 AOP 방법론을 통해서 웹 서비스 기능적 변화에 대한 구현을 보이고 있다. [7]의 연구에서는 인터넷 상에 흩어져 있는 수많은 웹 서비스들 중에서 어떻게 원하는 서비스를 효율적으로 탐색할 수 있는 가에 대해서 웹 서비스 매칭 알고리즘을 제안하고 온톨로지 개념을 적용하여 워크플로우 기법을 이용한 동적 웹 서비스 조합을 구현하였다.

그에 반해 정적인 방법은 실행 시간 전에 메타 데이터를 조합하는 방법으로 크게 두 가지 방식으로 나눌 수 있다. 첫째는 연산 호출에 대한 정적 선언을 읽고 미리 조합된 컴포넌트를 선언하고 이 컴포넌트의 인터페이스를 외부로 노출하는 방식이다. 두 번째는 웹 서비스 생성 도구를 통해서 기존에 개발된 웹 서비스 연산들을 조합하여 새로운 서비스 모델의 구조를 생성하는 방식이다. 여기에 속하는 [8]의 연구에서는 WS-BPEL만으로 서비스를 연결할 때 에러가 쉽게 발생할 수 있으므로 검증을 위해서 CPN(Colored Petri Net), 호환성 검사와 같은 자료 구조가 제안되었고, 이를 자동화하기 위해서 표준 혹은 전용의 언어로 매핑하는 연구가 진행되었다.

정적인 방법은 성능이 높지만 서비스 가용성과 유연성이 낮아질 수 있다. 이와 다르게 동적인 방법은 정적인 방법에 비해 약간 성능이 낮지만 서비스 가용성을 높일 수 있고 캐시를 이용한다면 정적인 방법보다 동일한 성능을 낼 수 있다. 그러므로 서비스가 로컬 서비스에는 정적인 조합을 사용하는 것이 좋다. 그러나 원격 서비스에는 동적인 조합을 사용하는 편이 낫다.

2. 웹 서비스 실행

웹 서비스는 기존 분산 기술에 비해 큰 성능 하락 없이 실행될 수 있다. 웹 서비스 실행 엔진은 일반적으로 요청 당 하나의 서비스 인스턴스를 생성하고 호출이 완료되면 해당 인스턴스를 소멸 또는 캐싱한다. 조합된 서비스를 실행하는 중간에 사용자와 상호작용을 해야 할 필요가 있을 경우 실행 엔진이 사용자의 컨텍스트 생성 방식과 파라미터를 전달 방식을 결정하는 것이 문제가 될 수 있다. 아울러 사용자와의 상호작용은 비지속적이라는 점을 고려할 때, 현재 수행하고 있는 오퍼레이션과 다음 오퍼레이션을 연결하기 위한 컨텍스트는 특정 퍼시스턴스 계층에 저장해야 한다. 그런 이유로 서비스 인스턴스가 소멸되더라도 트랜잭션을 유지하기 위해서는 클라이언트를 식별할 방법, 즉 세션의 개념이 요구된다. SOAP을 이용하는 대부분의 웹 서비스 클라이언트 프록시에서 HTTP 헤더 기반의 세션 ID를 사용할 수 있다[9]. [10]의 연구에서는 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. [11]에서 Choreography 선언을 통해서 명시되는 행위를 실행할 때 자원에 대한 접근 권한을 검증하는 알고리즘을 제시했다. W3C에서는 WS-CDL, WSCI과 같은 명세를 제안하고 있다. [12]에서는 XML 기반의 메시지 암호화, 디지털 시그니쳐 기능을 통합시키기 위해 XACML(eXtensible Access Control Markup Language) 기반의 웹 서비스 관리 서버를 제안하고 서비스 제공자에게 전달된 위임 확인을 위해 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.

III. 제안하는 기법

1. 웹 서비스 조합 및 실행 아키텍처

그림 1은 제안하는 웹 서비스 조합 및 실행 아키텍처를 나타낸다. 그림 1과 같이 웹 서비스 조합 및 실행 아키텍처는 크게 데이터 모델 계층(Data Model Layer, DML), 서비스 모델 계층(Service Model Layer, SML), 서비스 실행 계층(Service Execution Layer, SEL) 그리고 서비스 뷰 계층(Service View Layer, SVL)의 4계층으로 구성된다.

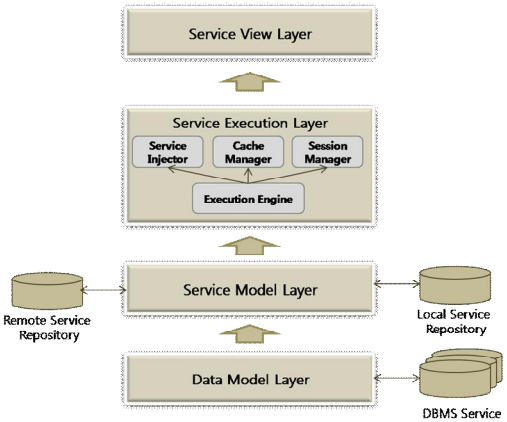

그림 1. 제안하는 웹 서비스 조합 및 실행 아키텍처
Fig. 1. Proposed Web Service Orchestration and Execution Architecture

DML 계층은 비즈니스 로직에서 사용되는 데이터와 서비스 사용자에게 제시할 뷰를 위한 모델링을 다루기 위한 서비스를 제공하는 계층이다. 대부분의 작업은 DML 계층 내부의 도메인 객체 인터페이스(Domain Object Interface, DOI) 통해 이루어진다. 여기서 DOI는 XML 기반의 도메인 객체에 대한 로딩, 검색, 수정, 저장 작업을 위한 인터페이스를 담고 있다.

SML 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. 웹 서비스 조합 및 실행 아키텍처는 검증된 서비스만을 런타임 저장소에 추가하기 때문에 서비스 인스턴스는 반드시 실행이 보장된다.

SEL 계층은 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 계층 내부에는 캐쉬 관리자, 세션 관리자, 서비스 주입자 서비스가 존재한다. 먼저, 캐쉬 관리자(Cache Manager)는 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. 세션 관리자(Session Manager)는 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. 서비스 주입자(Service Injector)는 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.

마지막으로 SVL 계층은 XML 기반의 메시지를 받게 된다. 그러나 일반적으로 웹 서비스 제공자는 서비스 실행 계층에서 넘겨진 결과를 GUI로 가공한다. 이를 위해 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. 웹 서비스 조합

2.1 비즈니스 로직 계층 조합

그림 2은 2개의 정적으로 정의된 서비스 인스턴스와 동적으로 생성된 프록시의 관계를 나타낸 클래스 다이어그램이다.

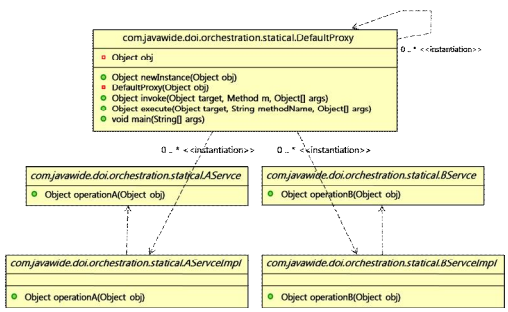

그림 2. 정적 서비스 클래스와 동적 프록시 간의 관계를 나타낸 클래스 다이어그램
Fig. 2. Class Diagram between Static Service Class and Dynamic Proxy

그림 3은 정적 서비스 인스턴스를 조합한 동적 프록시를 실행할 때의 결과이다. 정적 서비스 인스턴스를 조합할 경우 연산들은 그대로 존재하기 때문에 서비스 제공자는 서비스 인스턴스에 대한 실행을 2개의 컨텍스트로 나눠서 처리해야 한다. 이는 로컬에서 서비스 성능을 테스트할 때는 문제가 없지만, 원격에서 서비스 성능을 테스트할 때 2번의 네트워크 연결을 생성하기 때문에 개별 연산의 성능이 높더라도 네트워크 상황에 따라서 동적으로 동적 프록시를 생성한 방식보다 성능이 낮아진다.

```
before method operationA
Called OperationA
after method operationA
elapsed 0
before method operationB
Called OperationB
after method operationB
elapsed 0
```

그림 3. 정적 서비스의 조합 실행 결과
Fig. 3. Execution Result of Static Services

그림 4는 동적 서비스 클래스를 생성하기 위한 클래스의 선언이다. AbCodeDelegator는 appendCode()라는 추상 메서드를 가지고 있으며 추상 메서드를 상속하여 코드 바이트 순서를 명시하면 런타임에 클래스가 생성되어 동적 프록시를 통해서 객체를 생성할 수 있다.

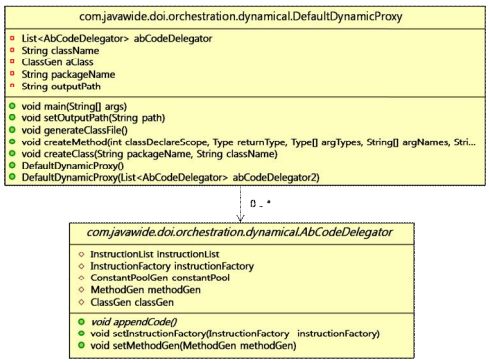

그림 4 동적 서비스 클래스의 생성을 위한 클래스 다이어그램
Fig. 4. Class Diagram for Generation of Dynamic Service Class

AbCodeDelegate 클래스의 instructionList 필드에는 실행 코드의 순서를 담는다. 실행 코드는 특정 오퍼레이션과 연관되어 지기 때문에 특정 메서드와 실행 코드를 연관시키기 위해서 Factory 클래스를 가진다. 실행 코드에서 사용되는 상수를 저장하기 위해서 Constant Pool을 선언하며, MethodGen은 하나의 메서드를 나타낸다. 여러 메서드는 하나의 ClassGen에 저장된다. 그림 5는 동적으로 클래스를 생성하는 예를 보인다. appendCode() 메서드에서 바이트 코드 순서대로 BCEL 라이브러리를 이용한다.

```
DefaultDynamicProxy proxy = new
DefaultDynamicProxy();
proxy.setOutputPath("build/classes");
proxy.createClass("com.javawide.doi.orchestrati
on.dynamical", "TestComposite");
proxy.createMethod(Constants.ACC_STATIC |
Constants.ACC_PUBLIC,
Type.VOID, new Type[] { Type.STRING }, new
String[] { "str" },
"appendString", new AbCodeDelegator() {
@Override
public void appendCode() {
...
}
});
proxy.generateClassFile();
```

그림 5. 동적 클래스 생성 예
Fig. 5. An Example for Creation of Dynamic Class

그림 6은 동적으로 생성된 바이트 코드를 보인다. 생성된 코드는 appendCode()에서 작성한 내용과 동일하게 작성된다. 이 때 생성된 바이트 코드는 JRE의 기본 명령 집합을 사용한다.

```
Compiled from "<DOKDO-WS Generated>
TestComposite.java"
public class com.javawide.doi.orchestration.
dynamical.TestComposite extends
java.lang.Object
  SourceFile: "<DOKDO-WS Generated>
TestComposite.java"
  minor version: 3
  major version: 45
  Constant pool:
const #1 = Asciz SourceFile;
const #2 = Asciz < D O K D O - W S
Generated>TestComposite.java;
const #3 = Asciz
com/javawide/doi/orchestration/dynamical/Tes
tComposite;
...
public static void
appendString(java.lang.String);
  Code:
   Stack=4, Locals=1, Args_size=1
   0: getstatic #12;
```

```
   3:  new #14;
   6:  dup
   7:  ldc #16;
   9:  invokespecial#20;
   12: aload_0
   13: invokevirtual#24;
   16: invokevirtual#28;
   19: invokevirtual#33;
   22: return
  LocalVariableTable:
   Start  Length  Slot  Name   Signature
   0             0             0         str
Ljava/lang/String;
...
생성자 Byte Code
}
```

그림 6. 동적으로 생성된 바이트 코드
Fig. 6. Generated Byte Code Dynamically

그림 7은 생성한 클래스를 동적 프록시로 로딩한 후 동적으로 추가한 메서드를 실행하는 것을 보인다.

```
Object target =
DefaultProxy.newInstance("com.javawide.doi.or
chestration.dynamical.TestComposite");
DefaultProxy.execute(target, "appendString",
new Object[] { "execute appendString" });
```

그림 7. 동적 프록시를 통한 동적 서비스 실행
Fig. 7. Dynamic Service Execution by Dynamic Proxy

그림 8에서 보이듯이 동적으로 생성한 서비스 인스턴스에 파라미터를 전달하여 appendCode()에서 생성하려고 했던 의도와 동일한 서비스 결과를 얻은 것을 확인할 수 있다.

```
append Method Value is execute appendString
```

그림 8. 동적 서비스 인스턴스 실행 결과
Fig. 8. Execution Result of Dynamic Service Instance

2.2 사용자 뷰 계층 조합

그림 9는 사용자 뷰 계층에서의 조합을 위한 코드의 예이다. 사용자 뷰가 로딩된 후에 조합을 시도하여 뷰 화면의 결과에 따라서 다른 조합된 결과를 보여준다. 실행되는 코드는 동적 함수 로딩을 통해서 수정될 수 있다.

```
Event.observe(window, "load", function(evt) {
Event.observe("orchestrate", "click",
function(evt) {
new Ajax.Request("StringA.html", {
onSuccess : function(response) {
  $("result").innerHTML += response.
responseText;
new Ajax.Request("StringB.jsp", {
onSuccess : function(response) {
  $("result").innerHTML += response.
responseText;
},
onFailure : function(response) {
  $("result").innerHTML += "fail";
}
});
},
onFailure : function(response) {
  $("result").innerHTML += "fail";
}});
});
});
```

그림 9. 동적 함수 로딩을 통한 조합
Fig. 9. Orchestration by Dynamic Function Loading

3. 조합 알고리즘

서비스는 서비스 실행 계층, 서비스 뷰 계층을 통해서 동적으로 조합될 수 있다. 그림 10은 서비스 실행 계층에서의 조합 알고리즘을 보인다. 서비스 실행 계층에서 전 후에 실행할 서비스에 대한 실행 코드를 추가한 동적 프록시를 서비스 인스턴스로 생성하여 서비스 인스턴스로 제공하게 된다. 동적 프록시는 AOP 방식을 이용해서 생성할 수도 있다.

```
   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 10. 서비스 실행 계층에서의 조합 알고리즘
Fig. 10. Orchestration Algorithm on Service Execution Layer

그림 11은 서비스 뷰에서의 조합 알고리즘을 보인다. 서비스 뷰 계층에서는 함수형 언어를 기반으로 함수들을 조합하여 서비스 그룹을 생성한다. 서비스 메타 모델 계층에서 서비스 프로토타입에는 성공과 실패에 대한 함수 그룹에 대한 선언이 없다. 그러나 함수를 제 1 클래스 객체(first-class object)로 사용하여 서비스 뷰 계층에서 동적으로 함수 그룹을 생성할 수 있다. 각 서비스 인스턴스에 대해서 성공 실패에 대한 함수 그룹은 서비스 실행 인스턴스로 동작하게 된다. 혹은 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 11. 서비스 뷰 계층에서의 조합 알고리즘
Fig. 11. Orchestration Algorithm on Service View Layer

서비스 실행 계층에서 생성되는 동적 프록시를 서비스 인스턴스로 제공하는 방식은 성능이 높고, 서비스 뷰 계층에서 접근할 때 단일 서비스로 인식되며 오류 검출이 쉽다는 장점이 있다. 그러나 사용하기 복잡하고 자원을 더 사용하게 된다. 이런 방식의 예는 관점 지향 프로그래밍이나 중간 언어 변환에서 찾아볼 수 있다. 프레임워크는 이런 방식의 사용을 지원한다. 이와 다르게 서비스 뷰 계층에서 사용하는 방법은 동적 프록시 보다 다소 낮고 서비스들은 다중의 서비스로 인식되며 오류 검출이 어렵다는 단점이 있다. 그러나 이해하기 쉽고 자원을 덜 사용하게 된다. Reflection을 통해 메타 데이터를 획득하고 WSDL을 생성하기 때문에 기존에 이미 존재하는 컴포넌트를 통해서도 웹 서비스를 쉽게 생성하고 이를 실행할 수 있다. 이 과정에서 별도의 추가적인 도구가 필요하지 않고, 기존에 존재하는 인프라 구조를 변경 없이 재사용할 수 있다. 서비스 실행 과정에서 컨텍스트 유지를 위해서 세션을 지원하며 동일한 서비스에 동일한 이름의 연산을 오버로드 할 수 있다. 또한 원격과 지역에 존재하는 서비스를 모두 호출할 수 있다. 또한 로컬 서비스 종점은 자동으로 생성되기 때문에 도메인 개발자는 서비스 종점에 대해서 전혀 고려하지 않아도 된다.

IV. 구현 및 실험

1. 구현 및 실험 환경

본 논문의 구현 및 실험 환경은 다음과 같다.
CPU : AMD Athlon 64 bit Dual Core 2.0 GHZ
운영체제(OS) : Windows XP Professional
개발언어 : Java(JDK 1.6.0 Update 10 버전)

특정 서비스의 요청은 HttpClient 4.0 Beta 1 버전을 사용하고 있으며 런타임에 동적으로 동적 프록시 및 서비스 인스턴스를 생성하기 위해 BCEL 5.2를 이용한다. 아울러 런타임에 동적으로 정적 프록시를 생성하기 위해 자바 API를 사용하고 있으며, view 계층 조합의 테스트를 위해서 prototype.js 라이브러리를 사용한다. 성능평가 척도는 도메인 프로세스 모델을 찾을 때 소요되는 평균 시간을 측정하였으며, 100 threads/sec 속도로 총 100,000번의 요청을 수행한다.

2. 실험 결과

그림 12는 비즈니스 로직 계층에서 조합된 서비스를 검색하고 초기화 하는 것에 대한 성능 평가 결과이다. 동일한 서비스 레지스트리를 통해서 접근하여 서비스 인스턴스에 대해서 JDBC, iBatis, Hibernate와 같은 다양한 ORM 방식을 이용해서 도메인 객체를 생성하는데 걸리는 시간을 나타낸다. 서비스 인스턴스에서 사용하는 퍼시스턴스 연산의 종류에 상

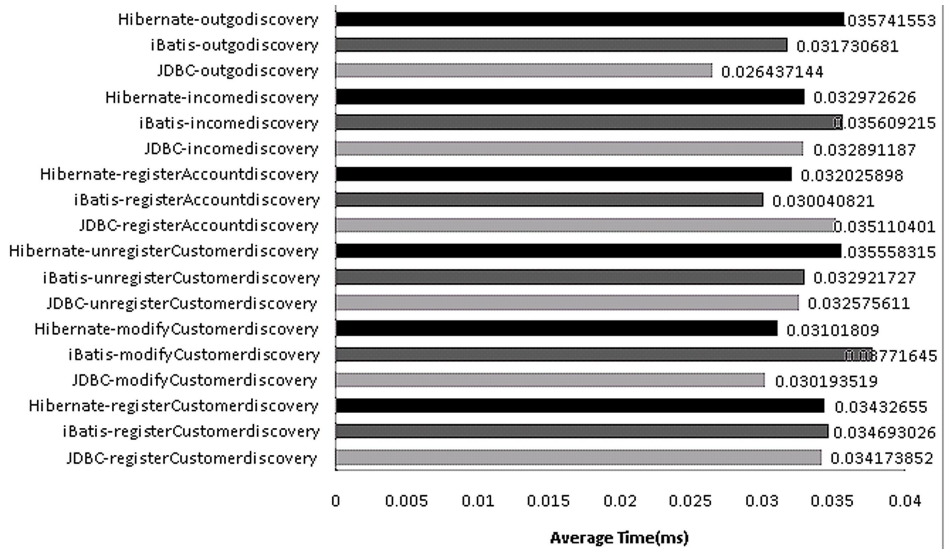

그림 12. 서비스 인스턴스 생성 시간 - 비즈니스 로직 계층 조합
Fig. 12. Creation Time for Service Instance - Business Logic Layer Orchestration

관없이 0.03 밀리초 정도의 시간에 서비스 인스턴스를 생성한다. 본 논문에서 제안하는 기법은 실행 시간 이전에 서비스의 연산의 조합을 완료하고 새로운 클래스를 생성한다. 따라서 실행 시간에 조합에 대한 어떠한 시간도 소요되지 않는다.

사용자 뷰 계층에서의 조합은 언제나 상수 시간에 종결된다. 서비스 연산을 묶은 함수 그룹에 대한 코드는 처리하는 프로그램 언어의 파서에 의해서 완전히 읽혀진 후에 단일 함수 Context를 생성하게 되며 그 후에 서비스 연산을 처리할 수 있는 상태가 된다. 이 과정은 단 한 번만 이뤄지며 실행 시간에는 생성된 함수 그룹 코드를 계속 이용하게 되므로 비즈니스 로직 계층에서의 조합과 마찬가지로 실행 시간에는 어떠한 시간도 소요되지 않는다.

3. 구현 결과

그림 13은 기본적으로 제공되는 웹 서비스 탐색기를 나타낸다. 본 논문에서 구현한 웹 서비스 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.

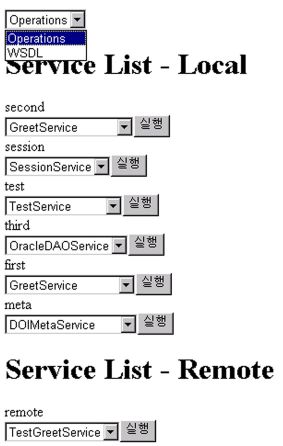

그림 13. 기본 웹 서비스 탐색기
Fig. 13. Basic Web Service Explorer

그림 14는 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.

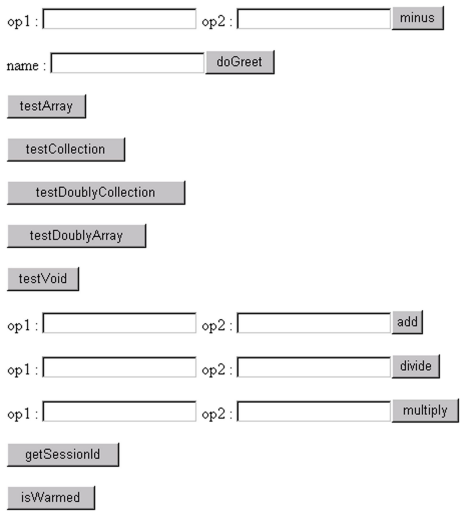

그림 14. 서비스 연산 결과 나열
Fig. 14. List of Service Operation Results

원격 서비스의 경우 그림 15와 동일한 폼을 생성하지만 실행할 때의 과정은 다르다. 그림 15는 원격 서비스의 실행 결과로 반환된 SOAP 메시지를 보인다. 원격 서비스는 WSDL 문서에서 연산을 추출하여 컨텍스트를 구성하고 SOAP 요청을 보낸다.

```
- <soapenv:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:doi="http://javawide.com/DOI">
  - <soapenv:Body>
    - <ns:minusResponse xmlns:ns="http://localhost/DOI">
        <ns:return> 1.0 </ns:return>
      </ns:minusResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

그림 15. 웹 서비스 실행(원격)
Fig. 15. Web Service Execution(Remote)

그림 16은 이기종 환경인 .NET 프레임워크에서 WSDL 문서를 참조하여 웹 서비스 프록시를 생성하는 것을 나타내며 자바를 이용하여 구현된 제안하는 기법을 통해 SOAP 메시지로 요청하여 처리된 결과를 나타낸다.

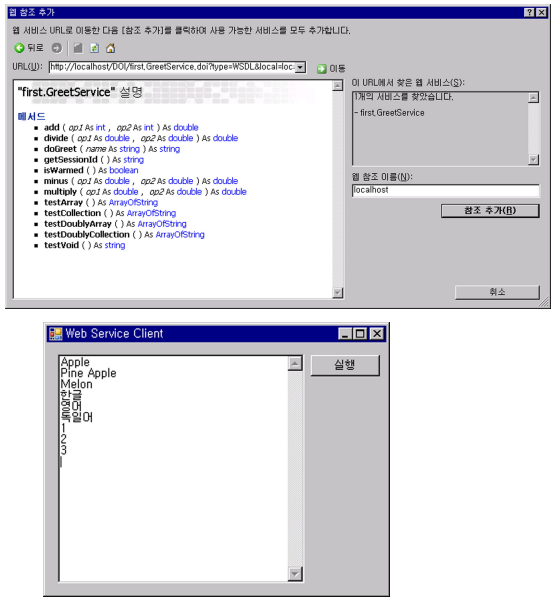

그림 16. .NET 웹 서비스 실행 결과
Fig. 16. Execution Results of .NET Web Service

V. 결론 및 향후연구

표준 웹 서비스 기술인 WS-BPEL이나 WS-CDL과 같은 언어를 기준으로 정적으로 조합하는 것은 높은 성능과 정확한 바인딩을 가능하게 한다. 그러나 단순히 IOCP를 기준으로 검색을 정확히 일치시키는 것은 시멘틱 웹 서비스 환경에 적합하지 않다. 본 논문에서는 이런 문제점을 해결하기 위해 동적으로 서비스를 조합하고 실행하기 위한 동적 조합 방식을 제안했다. 제안하는 기법은 동적인 특성을 얻기 위해서 서비스 실행 계층에서 AOP 기반의 접근법을 적용하고 Reflection 기술을 이용하여 메타 데이터를 획득하고 이를 기준으로 동적으로 실행이 가능한 프록시를 생성했다. 또한 서비스 뷰 계층에서는 함수 그룹을 통해 성공, 실패를 기준으로 처리되는 과정이 계속 결정되도록 구현했다. 웹 서비스 검색을 하기 전에 사용자는 도메인에 존재하는 서비스를 확인하고 각 서비스 별 성능을 확인하여 서비스를 선택할 수 있다. 조합된 웹 서비스는 세션을 유지한 상태로 트랜잭션 및 보안을 처리할 수 있으며, 선언 시간이 아닌 실행 시간에 동적 프록시를 조합하고 실행할 수 있는 방법을 제공한다.

향후 연구로는 웹 서비스 환경에서 사용자의 편리성 및 확장성을 제공함과 동시에 기존의 SOAP과 HTTP 요청, 응답,

그리고 실행 사이에서 상태를 유지하기 위한 세션을 지원하고 데이터 모델링과 반복되는 입력, 수정, 삭제, 검색의 퍼시스턴스 연산을 자동화 해 줄 수 있는 경량의 웹 서비스 프레임워크에 관한 연구가 필요하다.

참고문헌

〔1〕 Yanlong Zhai, Hongyi Su, and 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.

〔2〕 Freddy Lecue, Alexandre Delteil, and Alain Leger, "*Applying Abduction in Semantic Web Service Composition*," IEEE International Conference on Web Services, pp. 94-101, 2007.

〔3〕 Assia Ben Shil and 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.

〔4〕 선수림, 이금석, "AOP를 이용한 웹 애플리케이션의 보안성 강화 방안," 한국컴퓨터정보학회논문지, 제14권, 제2호, 119-128쪽, 2009년 2월.

〔5〕 Yongyan Zheng, and Paul Krause, "*Asynchronous Semantics and Anti-patterns for Interacting Web Services*," 6th International Conference on Quality Software, pp. 74-84, 2006.

〔6〕 Karthikeyan Ponnalagu, N.C. Narendra, Jayatheerthan Krishnamurthy, and R. Ramkumar, "*Aspect-oriented Approach for Non-functional Adaptation of Composite Web Services*," IEEE Congress on Services, pp. 284-291, 2007.

〔7〕 이용주, "시멘틱 e-워크플로우 프로세스를 이용한 동적 웹 서비스 조합," 한국컴퓨터정보학회논문지, 제10권, 제1호, 101-112쪽, 2005년 3월.

〔8〕 Hui Kang, Xiuli Yang, and Sinmiao Yuan, "*Modeling and Verification of Web Services Composition based on CPN*," IFIP International Conference on Network and Parallel Computing Workshops, pp. 613-617, 2007.

〔9〕 Costas Vassilakis, George Lepouras, and Akrivi Katifori, "*Web Service Execution Streamlining*," International Conference on Service Systems and Service Management, Vol. 2, pp. 1564-1569, 2006.

〔10〕 Feng Liu, Gesan Wang, Li Li, and Wu Chou, "*Web Service for Distributed Communication Systems*," IEEE International Conference on Service Operations and Logistics, and Informatics, pp. 1030-1035, 2006.

〔11〕 Federica Paci, Mourad Ouzzani, and Massimo Mecella, "*Verification of Access Control Requirements in Web Services Choreography*," IEEE International Conference on Service Computing, Vol. 1, pp. 5-12, 2008.

〔12〕 Hyun Sik Hwang, Hyuk Jin Ko, Kyu Il Kim, and 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.

저 자 소 개

임 은 천

2007년 2월
순천대학교 정보통신공학부 공학사
2009년 2월
순천대학교 멀티미디어공학과 공학석사
관심분야 : 멀티미디어 데이터베이스, 지능정보시스템, 웹 서비스, RFID/USN

심 춘 보

2003년 2월
전북대학교 컴퓨터공학과 공학박사
2005년 2월 ~ 현재
순천대학교 정보통신공학부 교수
관심분야 : 멀티미디어 데이터베이스 & 정보검색, 웹기반 지능정보시스템, RFID/USN