SMFD 기반의 SOA 비즈니스 서비스와 어플리케이션 서비스 연결 테스트 설계☆

A Test Design for Service Connection of Business Service and Application Service in SOA using Service Message Flow Diagram

지 은 미* 윤 회 진** 최 병 주***
Eun Mi Ji Hoijin Yoon Byoungju Choi

요 약

서비스 기반 아키텍처의 서비스들 사이에서 연결 테스트를 하기 위하여 서비스 기반 아키텍처의 연결 특성을 분석하고, 분석을 통해서 서비스 기반 아키텍처의 특성을 표현한 서비스 메시지 흐름도를 정의한다. 이 정의를 토대로 본 논문은 서비스 연결 테스트하기 위한 방안을 제안하고, 실제 서비스 기반 아키텍처를 사용하는 회사에 이를 적용한다. 최근 많은 기업들에서 기존의 시스템을 서비스 기반 아키텍처로 재구축하려는 움직임이 활발하다. 그러나 이러한 프로젝트들이 현실적으로 실제 시스템으로 사용되기 어려운 점은 그에 대한 연결테스트의 부재이기 때문이다. 또한 SOA에서의 통합과 연결에 차이에 대해 느끼지 못한다는 문제와 그에 따른 서비스의 계층 간의 연결에 대한 테스트의 필요성을 느끼지 못한다는 문제이다. 따라서 본 논문에서는 서비스 기반 아키텍처가 준수해야 하는 계층구조를 기반으로 서비스 연결 테스트 방안을 제안하고, 차별화되는 두 가지 사례에 적용한다.

Abstract

This paper defines a diagram showing the characteristics of service connections in SOA, and it designs the connection testing with the diagram. The detail steps of building test requirements in the test design phase are explained, and the steps are applied to two different enterprise cases as examples. The cases are real SOA system of real enterprises. Recently, many enterprises have been trying to reconfigure their system as SOA systems. However, these projects are hard to be applied to their mission-critical real systems, since there is few testing technique understanding SOA characteristics. One of the missing characteristics of SOA in testing is that SOA needs a layering systems between business processes and applications. This paper focuses on the layering system of SOA to solve the testing problem in SOA.

☞ keywords : Service Oriented Architecture, Service connection, Testing

1. 서 론

기업의 IT 환경이 거대해지고 복잡해짐에 따라 비즈니스에 대한 낮은 대응력, 고비용 구조, 다양한 시스템통합 문제 등 심각한 문제에 직면해 있다. 이에 따른 해결방안으로 제시된 것이 SOA(Service Oriented Architecture)[1]이다. 복잡하고 급변하는 비즈니스 환경과 다양한 사용자의 요구에 유연하게 대응하기 위해 비즈니스 컴포넌트를 서비스화 하고 이를 계층적 구조 중심으로 통합•재구성 하는 것으로 소프트웨어 재사용성과 단위 기능들 간의 상호 호환성에 중심을 둔 소프트웨어 설계방식이다. SOA는 느슨한 연결(loosely coupled)[2]를 대표적인

* 준 회 원 : 이화여자대학교 컴퓨터학과 석사과정
jjiem@naver.com
** 정 회 원 : 협성대학교 컴퓨터공학과 교수(교신저자)
hjyoon@uhs.ac.kr
*** 정 회 원 : 이화여자대학교 컴퓨터학과 교수
bjchoi@ewha.ac.kr
☆ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2007-331-D00406)
[2008/05/09 투고 - 2008/05/18 심사 - 2008/09/25 심사완료]

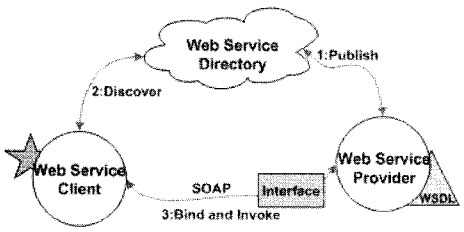
특징으로 하여 비즈니스 프로세스와 IT기술 사이의 직접적인 연결이 아닌 추상적인 연결을 구현 할 수 있도록 한다. 이를 통하여 비즈니스 프로세스의 변화를 시스템에 보다 간단하게 적용할 수 있다. SOA 도입으로 인한 주요 효과는 통합(integration)이 갖는 오래된 문제를 해결하는 것이다. 여러 분석에 따르면, 전형적인 IT예산의 최고 30%가 통합 활동에 할당되고 있다[3]. 이때 말하는 통합이라는 것은 프로세스 통합, 기업체 통합, 그리고 M&A(Mergers and Acquisitions) 통합 등을 포함한다. 만일 이러한 통합에 쓰일 비용이 조금 줄어든다면, 남은 예산을 프로젝트의 전략을 위한 관리에 쓸 수 있게 된다. 기업의 사업 환경이 점점 복잡해지고 변화함에 따라 기업의 계속적인 성장을 위해서 업무 절차나 흐름을 수시로 변화시킬 수 있는 융통성 있는 체계를 갖추어야 한다. 많은 기업들이 이런 요구사항을 충족해줄 수 있는 SOA로 기존의 시스템을 재구축하거나 새로운 기능을 SOA 기반으로 구축하려는 움직임이 활발하다. 하지만 기대와는 달리 파이럿 프로젝트들은 연이어 본 프로젝트로 연결되지 못하고 있고 파일럿 프로젝트가 성공을 했다고 하더라도 다양한 연결 테스트가 빠져 있다면 실제 효과에 대한 확신을 갖기가 어렵다. 왜냐하면 HP 소프트웨어 유니버스 2007[4]에서 지적되었던 서비스 연결 테스트 부재 때문이다. 기존의 웹 서비스의 SOAP 메시지를 중심으로 이루어지는 테스트 방안과 다른 관점으로, 본 논문은 SOA의 서비스 연결 개념을 정의하고 서비스들의 연결구조들을 분석하여 같은 계층 내에서의 직접 연결이 아닌 계층 간의 연결로 비즈니스 프로세스 계층과 어플리케이션 계층 사이의 연결 테스트 요구사항을 추출 한다. 또한 요구사항을 표현하기 위해서 메시지 흐름도 기반의 다이어그램을 정의한다. 따라서 서비스 메시지 흐름도(SMFD : Service Message Flow Diagram) 기반의 SOA 서비스 연결 테스트 설계를 하고, 이상적인 SOA를 하기 위해서는 느슨한 연결을 유지하는 계층 구조에서 서비스 연결을 분석하여 연결 테스트 요구사항을 추출하여야 하며 연결 테스트 항목 추출 방법과 연결 테스트가 필요함을 증명한다.

본 논문은 서론에 이어 2장 관련연구에서, 웹 서비스와 SOA의 개념과 관계에 대해 소개하고 서비스 연결 특성과 서비스 간의 연결 방식을 설명한다. 3장은 서비스 기반의 계층구조와 SOA 연결을 분석하여, '서비스 통합(Integration)'과 '서비스 연결(Connection)' 특성을 구별하고 특성을 기반으로 하여 SMFD를 정의하고 테스트 요구사항을 추출한다. 이는 SOA 서비스 연결 테스트 설계 절차에 맞추어 기술한다. 4장은 차별화되는 두 가지의 실제 SOA 사례에 서비스 연결 테스트 설계 절차를 적용한다. 5장에서 결론과 향후 연구 방향을 기술한다.

2. 관련연구

2.1 웹 서비스와 SOA

웹의 이용이 급속하게 증가함에 따라, 새로운 분산 플랫폼을 기존 환경들과 효율적으로 통합할 수 있는 방안이 요구되었으며 그 해결방안으로 웹 서비스가 등장하였다[5]. 웹 서비스는 인터넷 네트워크를 통해 다수의 비즈니스 시스템들을 표준화된 기술로 결합시켜 고객이 원하는 정보나 응용 기능을 구현하는 기술로, 웹 전송 프로토콜을 통해 XML 기반의 메시지를 주고받으면서 특정 작업을 수행하는 것이다. 이와 같은 정의를 통해서 매우 유연하게 프로그램 언어 간의 의사소통 차이를 극복해 주는 역할을 하므로 웹 서비스는 이상적인 SOA를 구현할 수 있는 수단으로 인식되고 있다[6]. 웹 서비스는 UDDI(Universal Description, Discovery and Integration), WSDL(Web Service Description Language), SOAP(Simple Object Access Protocol) 이라는 XML 기반의 세 가지 공개 표준[7]을 이용하여, 웹을 통해 XML 메시지를 전송함으로써 이 기종 시스템간의 상호작용을 돕는 소프트웨어 시스템으로 플랫폼이나 벤더에 무관한 개방형 표준을 지향한다. 웹 서비스를 구성하는 각 구성요소는 서로 상호 관계를 가지며 웹 서비스 배포는 다음 그림1과 같다.



(그림 1) 웹 서비스 배포

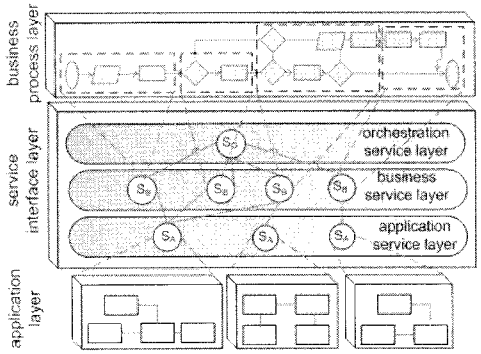
SOA는 웹과 같은 네트워크에서 가능한 서비스를 사용하는 소프트웨어 어플리케이션을 구축하기 위한 아키텍처 스타일로 재사용이 가능하도록 소프트웨어 컴포넌트 사이에 느슨한 연결을 활성화 시킨다. SOA가 갖추어야 할 일반적인 원리로 느슨한 연결을 기초로 상호작용하도록 설계되어야 하며, 서비스는 표준화된 방법을 통해 모든 환경에서 호출이 가능해야 한다. 이것은 서비스 호출 메커니즘이 널리 채택된 표준에 근거해야 함으로 플랫폼에 독립적이어야 하고 서비스는 서비스의 정의와 위치 정보를 UDDI와 같은 저장소에 저장하고 여러 클라이언트를 통해 서비스의 위치와 상관없이 등록, 호출 될 수 있는 위치 투명성을 지원해야 한다[8]. SOA 서비스 집합은 기업의 비즈니스 로직과 기술 특화된 어플리케이션 로직으로 표현되며, 이렇게 표현될 때 각 영역은 다른 영역들에 직접적으로 의존하지 않고 독립적으로 존재하게 된다. 또한 민첩한 SOA를 구축하기 위해서는 각 서비스가 그 서비스를 처리하는 로직에 대한 의존성을 최소화하는 것이므로, 비즈니스 서비스 계층과 어플리케이션 서비스 계층 위에 상위 컨트롤러인 오케스트레이션 서비스를 구축한다. SOA의 장점으로는 서비스는 구현 기술에 관계없이 잘 정의된 인터페이스를 가진 소프트웨어 컴포넌트이고 서비스 인터페이스를 구현으로부터 분리시켰다는 것이다. 즉, 서비스라 불리는 분할된 어플리케이션 조각들을 단위로 느슨한 연결을 통하여 하나의 완성된 어플리케이션을 개발하기 위한 소프트웨어 아키텍처라 할 수 있다[9].

앞에서 말했듯, 웹 서비스는 SOA를 구현하기 위한 하나의 해결 방안이다. 웹 서비스의 WSDL은 표준화된 방식으로 웹 서비스의 인터페이스를 기술하는 XML 기술이고, SOAP은 원격 응용프로그램 간의 정보를 교환하는데 필요한 구조를 표준화한 것이다. UDDI는 웹 서비스의 공개와 검색을 위한 XML 레지스트리의 구현과 사용방법을 표준화한 것으로 스펙에는 웹 서비스 중개자가 운영해야 할 XML 레지스트리 기능과, 웹 서비스 제공자가 어떻게 자사의 웹 서비스를 공개해야 되는지, 그리고 웹 서비스 소비자가 어떻게 검색할 수 있는지 대해서 정의하고 있다. SOA가 웹 서비스와 같은 특정 기술을 그 자체로 언급하거나 필요로 하고 있지 않지만, SOA의 느슨한 연결을 뒷받침할 수 있는 보편화된 기술은 현실적으로 웹 서비스가 가장 적합하다.

2.2 서비스 연결 특성

SOA는 계층을 유지함으로써, 비즈니스 프로세스의 변화는 비즈니스 서비스에 반영되고, 기술적인 현상의 변화하는 어플리케이션 서비스에 반영된다. 따라서 그림2와 같은 계층구조에 배치해야 하며, 각 계층에 속하는 서비스들은 자신과 같은 계층과의 직접 연결이 아닌 상위와 하위 계층의 서비스연결을 유지하도록 해야 한다[10].

서비스 단위 테스트는 기존의 구현 코드 기반

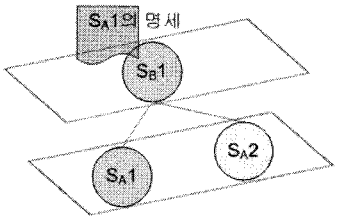


(그림 2) SOA의 계층 구조와 서비스들의 배치[11]

또는 블랙박스 테스트 방법으로 접근 가능하나, 서비스들의 연결 테스트는 기존의 방법 적용과 더불어 SOA의 서비스 연결 개념을 정확하게 반영해야 한다. 여기서 서비스 연결 테스트란, 서비스 지향 아키텍처의 계층적 구조 중심으로 같은 계층의 직접 연결이 아닌 서로 다른 계층에서 서비스가 서로의 필요성에 의해 연결을 테스트 하는 것을 말한다. 서비스 연결 테스트에 대한 서로 다른 계층에서의 서비스 관계 분석이 필요하며 이때 발생하는 오류에 대한 테스트 방안이 특별히 요구되므로, 테스트 요구사항이 필요하다. 그림 2를 보면, 어플리케이션 계층, 서비스 인터페이스 계층, 비즈니스 프로세스 계층으로 나뉘며, 서비스 인터페이스 계층은 비즈니스 프로세스 계층과 어플리케이션 계층 사이에 위치한다. 이 서비스 인터페이스 계층은 서비스 연결이 발생하는 곳이고, 따라서 SOA의 특성이 가장 잘 나타나는 곳이다. 비즈니스 서비스는 비즈니스 프로세스를 이루는 작업들로 이루어지며, 이는 해당 작업을 수행하기 위하여 어플리케이션 서비스에게 요청 메시지를 전송하게 된다. 어플리케이션 서비스는 비즈니스 프로세스 또는 그의 작업과는 전혀 무관하며 단지 어플리케이션들이 갖는 오퍼레이션들을 모아서 구성한 서비스 단위이다.

비즈니스 서비스로부터 요청 메시지를 받게 되면 어플리케이션 서비스는 그와 연결된 오퍼레이션의 구현 객체는 또 다른 제 3의 장소에 있을 수 있다. 그렇다면 이때도 물론 SOAP을 통한 메시지 전송으로 요청 메시지가 전송되며, 수행된 결과가 응답 메시지로 되어 해당 어플리케이션 서비스에게 보내진다.

요구사항으로 들어온 비즈니스 프로세스 명세를 분석하여 후보서비스(Service Candidate)들로 분할하고 이 하나의 후보 서비스를 구현하기 위한 설계를 서비스 단위로 하게 된다. 이 설계 과정에서 비즈니스 프로세스 관점 정의된 서비스로 비즈니스 요구사항을 관련 제약사항 의존 관계, 외부 영향요소를 가진 비즈니스 서비스(Business Service) S_B를 구현하고, 이를 기술적으로 구현하는 어플리케이션 서비스(Application Service) S_A, 들로 각 비즈니스 서비스들과 연결된다. 이때 전체 프로세스를 반영하여 서비스들의 흐름을 제어하는 서비스가 오케스트레이션, 즉 오케스트레이션(Orchestration Service) So, 이고 이는 비즈니스 프로세스를 일련의 비즈니스 서비스들로 표현하여, 서비스들 사이의 흐름을 제어한다. 즉, 계층 구조는 비즈니스 요구사항 관점과 기술적 관점의 두 방향에서의 추상화 정도에 따라 나눠지며 상단에 이 두 계층 간에 서비스들을 조정하는 오케스트레이션 계층이 존재한다. 각 계층은 계층 별로 특정 이슈들을 해결 할 수 있게 각 계층은 특화된 관점에 따라 추상화를 달리한다. SOA 내부에서, 서비스는 다른 서비스나 프로그램에 의해 사용될 수 있다. 서비스들 간의 연결은 상호작용에 기반하고 있으므로 서비스 상호 간에 식별이 가능해야 하고 이는 서비스 명세를 통해 가능하다. 다음의 그림 3은 간략한 서비스 간의 연결의 모습을 보여주고 있는데, 서비스 S_A1은 서비스 S_A1의 서비스 명세에 접근하여 서비스 S_A1과 통신하는데 필요한 모든 정보를 얻을 수 있다.



(그림 3) 서비스 간의 연결

서비스는 이렇게 서비스 명세를 사용하여 결과적으로 느슨한 연결을 형성하게 되고 다시 말해, 서비스 S_B1이 서비스 S_A1의 서비스 명세를 갖고 있기 때문에 서비스 S_A1을 식별 할 수 있는 것이다. 서비스가 상호작용하여 의미 있는 것을 얻기 위해서는 서비스 간에 정보를 교환해야만 한다. 그러므로 이러한 서비스들이 느슨하게 연결되도록 하기 위해서는 커뮤니케이션 프레임워크가 필요하다. 이와 같은 프레임워크 중의 하나가 메시지이다. 서

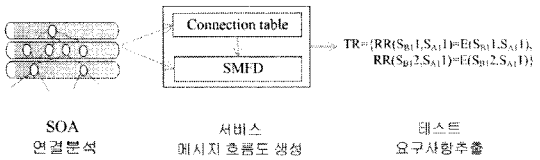
비스 간의 커뮤니케이션 방식은 서비스가 메시지를 전송하고 나면, 그 이후 메시지에 대한 통제력은 상실하게 된다. 그러므로 메시지는 커뮤니케이션의 독립적 단위로서 존재해야만 한다. 이는 메시지도 서비스처럼 자율적이어야 한다는 것을 의미하고 그 결과, 메시지는 프로세스 로직의 일부를 스스로 통제 할 수 있는 충분히 지능적인 도구가 될 수 있다. 또한 메시지는 SOAP 헤더를 사용함으로써 서비스 재사용을 간접적으로 지원하게 된다.

2.3 웹 서비스 테스트 도구

SOA는 서비스들이 조합되고 공유될 수 있는 장점이 있지만, 이는 만약 배포된 서비스 중 하나가 기능상의 문제가 생긴다면 이 서비스를 공유하고 있는 여러 어플리케이션에도 문제가 생김을 의미한다. 이러한 이유로 SoapUI[12], JUint[13], TestMaker[14], WebInject[15]와 같은 서비스 테스트 도구들이 개발 되었다. 웹 서비스 테스트 도구들은 단위 테스트와 SOAP메시지에 대한 테스트를 해주므로 테스터들을 편하게 하고 있다. 하지만, 일반 웹 테스트와 달리 SOA의 테스트는 SOA 특성을 고려하여 단위 테스트와 SOAP메시지에 대한 테스트를 해야 이상적인 SOA를 설계 할 수 있을 것이다. 일반적인 SOA개발자들은 오케스트레이션 서비스를 대상으로 테스트 도구를 돌리면 전체적인 연결테스트가 되는 것으로 쉽게 생각할 수 있다. 따라서 단위테스트 도구로도 충분히 테스트가 가능하다고 믿고 있고, 테스트를 성공적으로 실행 했다고 믿지만 웹 서비스 테스트 도구가 서비스들을 테스트 하도록 환경을 제공했을 뿐 계층 간의 서비스 연결 테스트를 해주지 못하는 실정이다. SOA의 규칙들을 준수할 때, SOA가 주는 진정한 이득을 얻을 수 있다. 이렇듯 SOA의 이득을 얻기 위해서는 연결테스트를 해야 하지만, 웹 서비스 테스트 도구 SoapUI, JUnit, TestMaker, WebInject 로는 환경만을 제공할 뿐 테스트의 요구사항과 계층 간의 연결 분석이 필요하며, 이를 정의하였을 때 효율적인 연결 테스트를 기대할 할 수 있다.

3. SOA 서비스 연결 테스트 설계 절차

SOA 서비스 연결 테스트하기 위해서는 그림4의 절차로 구성된다.



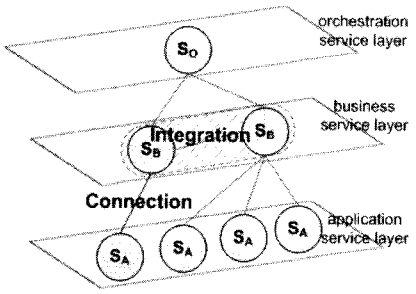
(그림 4) 서비스 연결 테스트 절차

SOA의 연결을 분석하고 서비스 메시지 흐름도를 정의한다. SOA 특징을 준수하는 연결 특성을 표현하기 위한 다이어그램은 서비스의 연결 테스트 요구사항을 생성하기 위함이다[18]. SMFD은 서비스 연결 특성으로 만들어진 연결테이블(connection table)을 토대로 정의하고 테스트 요구사항을 추출하므로 SOA 서비스 연결 테스트 설계를 할 수 있다.

3.1 SOA 연결 분석

SOA 시스템 구축에 대한 여러 방법들이 상향식 또는 하향식 접근 등으로 제안되고 있다. 이 두 가지 방법은 아래의 그림 5와 같이 모두 비즈니스 서비스 계층과 어플리케이션 서비스 계층을 구성하며, 이들 계층에 오는 서비스들은 계층 간의 특성에 맞게 구현되어야 한다. 계층 간의 특성으로는 비즈니스 서비스는 비즈니스 프로세스를 이루는 작업들로 이루어지며 이는 해당 작업을 수행하기 위하여 어플리케이션 서비스에게 요청 메시지를 전송하게 된다. 그리고 어플리케이션 서비스는 비즈니스 프로세스 또는 그의 작업과는 전혀 무관하며 단지 어플리케이션들이 갖는 오퍼레이션(Operation)

들을 모아서 구성한 서비스 단위이다. 비즈니스 서비스로부터 요청 메시지를 받게 되면 어플리케이션 서비스는 그와 연결된 오퍼레이션을 수행한다. 따라서 하나의 트랜잭션에 대하여 비즈니스 서비스가 어플리케이션 서비스에게 요청메시지를 보내고 어플리케이션 서비스가 하위계층의 어플리케이션들로부터 결과를 받아 비즈니스 서비스에게 응답메시지를 보내는 연결 구조가 존재한다.



(그림 5) SOA 서비스 연결과 통합

요구인터페이스와 제공인터페이스를 통한 통합으로 이루어지는 기존의 CBSD와 다르게, SOA는 "no integration"을 원칙으로 한다. 즉 서비스들의 인터페이스들 사이의 직접적인 물리적 연결을 시도하지 않고, 서비스들 사이에 메시지 전송을 통한 요청-응답 메카니즘을 통한 연결이 이루어진다. SOA의 '서비스 연결'은 '서비스들의 통합'과는 다른 관점을 갖는다. 자세히 설명하면, 약 결합 형태의 서비스를 지향하기 위해서는 서비스 간 결합도가 낮아야 한다. 서비스 간 결합도가 높으면 하나의 서비스로 통합이 되어야 한다. 결합도가 높으면 모듈의 변화에 따른 다른 모듈로의 파장이 커지게 되고, 또한 같은 계층에서 연결이 이루어진다면 유사한 서비스의 역할을 하는 것이라고 여기며 통합이 이루어져야 한다. 그에 반해 서비스 연결은 앞에서 말했듯 서비스 지향 아키텍처의 계층적 구조 중심으로 같은 계층의 직접 연결이 아닌 서로 다른 계층에서 서비스가 서로의 필요성에 의해 연결이 되는 것을 말한다. 따라서 서비스의 통합 테스트와 서비스 연결 테스트에 대한 서로 다른 계층에서의 서비스 관계 분석이 필요하다. 이때 발생하는 오류에 대한 테스트 방안이 특별히 요구되므로, 테스트 요구사항이 필요하다.

SOA의 서비스 연결 특성이 주는 이점을 활용하기 위해서는 SOA가 추구하는 연결 특성을 잘 따라야 한다. SOA 서비스 연결이 갖는 다섯 개의 특성은 다음과 같다.

(1) 요청한 서비스에게로 요청 결과를 보내야 한다. 요청 메시지 흐름에 대한 응답으로서의 역방향이 반드시 존재해야 한다. (2)하나의 트랜잭션은 So로 시작하여 So로 끝난다. 연결 메시지의 흐름의 초기 요청은 사용자에 의해 접근되는 So에서 출발하여, S_B를 거쳐 S_A에게 전달되고, 그를 수행한 결과를 응답으로서 거꾸로 S_B를 거쳐 So에게 전달하게 된다. (3)비즈니스 서비스는 비즈니스 프로세스 계층을 반영하고 어플리케이션 서비스는 어플리케이션 계층을 표현해야 한다. 이를 통하여 비즈니스 프로세스가 변경되면 비즈니스 서비스에 그 변경이 반영되고, 어플리케이션의 수정이 요구되면 그를 호출하고 있는 어플리케이션 서비스에 영향을 미치게 된다. (4) 어플리케이션 서비스 계층과 비즈니스 서비스 계층의 다중 계층일수 있다. 비즈니스 서비스 계층의 경우 그 종류에 따라 상위에 태스크 중심 비즈니스 서비스가 존재하며 하위에 엔트리 중심 비즈니스 서비스가 존재할 수 있다. 어플리케이션 서비스 층의 경우도 상하 관계의 호출이 이루어질 수 있다. (5) 같은 계층 내에서의 직접 연결은 지양해야 한다. 이는 서비스들 사이의 통합을 유발하게 되므로 SOA가 지향하는 연결 구조가 아니다.

3.2 서비스 메시지 흐름도

서비스 메시지 흐름도 (Service Message Flow Diagram : SMFD)를 다음과 같이 정의하고, 정의2는 서비스 연결을 위한 테스트 요구사항을 정의한다.

정의 **1: SMFD**는 비 순환 그래프, **G(V,E)**이다.

- V 는 오케스트레이션 서비스, 비즈니스 서비

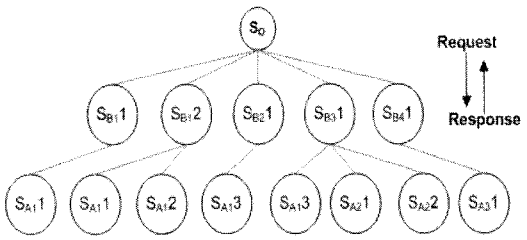
스의 작업 혹은 어플리케이션 서비스의 오퍼레이션을 표현하고 있다.

- E 는 부모가 자식의 서비스를 요청한 메시지 흐름을 표현한다.
- (p,q) ∈ E, where p V is the parent of q∈V.
- V 는 서비스의 정점의 오퍼레이션을 표현하는 이름을 가진다.

정의 **2:** 서비스 연결을 위한 테스트 요구사항 **(TR)**

- SMFD의 선은 SOA에서 검증된 연결로 표현된다.
- 선을 통한 응답 값과 예상출력 값이 같은지 연결 테스트로 체크해야 한다.
- 트랜잭션은 오케스트레이션 서비스에서 출발한 요청메시지에 대한 응답메시지가 오케스트레이션 서비스로 도착하는 일련의 메시지 흐름을 의미한다.

서비스는 공개표준을 기반으로 하여 메시지를 전달하게 되며, 메시지를 통해 연결이 된다. 그 연결은 웹 서비스와 이를 둘러싼 환경에서 다양한 SOAP 메시지를 처리한다. 메시지 처리 로직은 WSDL 처리와 관련된 서비스 로직뿐만 아니라 실행 서비스와 서비스 에이전트 간의 조합도 이에 해당한다. SMFD는 SOA의 특성을 표현하기 위하여 메시지 메커니즘과 서비스 계층별 특성을 나타내기 위한 구조를 갖는다. 메시지 메커니즘의 표현을 위하여 기본 선 외에 쓰레드를 이용하여 요청 메시지 흐름은 위에서 아래 방향으로 그리고 응답 메시지는 아래에서 위 방향으로 그리도록 명시한다. 각 노드는 서비스를 표현하고 선은 서비스 호출을 위한 메시지 흐름을 표현한다. 노드들은 서비스 수행에 대한 응답 메시지 흐름을 표현하는 쓰레드를 가질 수 있다. 각 계층의 서비스 특성을 표현하기 위해서는 계층별 그래프의 노드들은 서로 다른 내용을 표현하고 있다. 비즈니스 서비스는 비즈니스 프로세스의 작업을 표현하는 서비스로 작업을 처리하며, 작업의 엮은 기준에 따라 태스크 중심 서비스와 엔티티 중심 서비스로 나눠진다.



(그림 6) SMFD

먼저, 오퍼레이션에 의해 서비스를 분석하고 각각의 작업과 오퍼레이션을 위한 노드를 그린다. 비즈니스 서비스는 비즈니스 프로세스를 포함한 작업들로 연관 되고 어플리케이션 서비스는 오퍼레이션과 연관된다. 작업을 위한 노드들은 SMFD에서 오퍼레이션을 위한 노드들 보다 상위레벨에 위치해야 한다. 그리고 표현의 편의상 $S_A1..n$으로 서비스를 표현하고, 그에 속하는 작업 또는 오퍼레이션에 순차적인 번호를 붙이고, 노드들은 자신만의 이름을 가진다. 예를 들어, 어플리케이션 서비스 S_A1의 3번째 오퍼레이션은 $S_{A1}3$이라고 불린다.

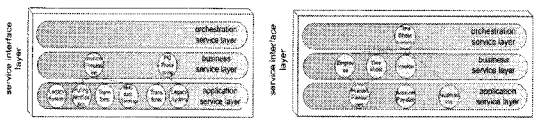
3.3 테스트 요구사항 추출

테스트 요구사항이란 테스트 케이스를 만족시키고 커버할 수 있는 소프트웨어 산물의 명확한 요소이다[16]. 테스트 요구사항들은 주로 집합들로 요구되며, 요구사항의 집합을 TR로 칭하며, TR은 소스코드, 디자인 컴포넌트, 모델링 요소의 명세서, 또는 입력공간의 기술을 포함한 SW 산물의 다양한 부분을 묘사할 수 있다. 서비스끼리의 연결들이 복잡할수록 많은 트랜잭션을 요구하게 되며, 그에 따른 테스트 요구사항의 수도 늘어나게 된다[17]. 그 결과 연결 테스트가 커버해야 하는 테스트 요구사항의 수도 늘어나게 된다. 몇몇의 트랜잭션이 테스트 요구사항의 크기를 결정하며 나아가 테스트 비

용에 영향을 미친다. 따라서 SMFD에서 나올 수 있는 의미 있는 트랜잭션들을 연결 테스트 데이터가 커버할 수 있어야 한다. 테스트 요구사항은 모든 연결을 포함하기 위해 필요하며 연결은 서비스 사이의 메시지 흐름으로 만들어진다. 오퍼레이션 간이나 작업 간이 아닌, 비즈니스 서비스의 작업은 어플리케이션 서비스의 오퍼레이션에게 요청메시지를 보내고, 그 후 오퍼레이션은 작업에게 응답 메시지를 되돌려 보낸다. 그러므로 서비스 지향 아키텍처에서의 연결은 작업과 오퍼레이션의 쌍으로 결정된다. 현재 서비스 단위의 연결은 묘사되어 있지만 테스트 데이터에 의해 포함되어야 하는 메시지 흐름을 알기에는 어려울 수도 있다. 연결에 관한 정보를 얻기 위해 서비스 단위의 연결보다 더 많은 것이 필요하다. 즉, 이전에 정의한 SMFD가 필요하다. 모든 연결은 SMFD에서 다른 계층을 교차하여 그려진다. 이는 오퍼레이션끼리 요청-응답 관계를 가지지 않으며, 작업 또한 서로 그러한 관계를 가지지 않음을 의미한다. 이는 오퍼레이션 간의 연결, 작업간의 연결은 서비스의 강한 결합을 야기할 수 있기에 권하지 않는다. 다음 장은 SMFD를 통한 테스트 요구사항 추출 과정을 실제 SOA을 구축한 회사에 적용할 것이다. 이 과정을 통하여 생성된 테스트 요구사항들을 만족시키기 위한 테스트 데이터 선정 기준이 추후 필요하다. 이는 기존의 웹 서비스의 SOAP 메시지에 기반을 둔 현재의 테스트 데이터 선정[18, 19, 20] 방안 등이 이 부분에 적용될 수 있다.

4. SOA 어플리케이션에서의 적용

서비스 연결 분석을 통해 비즈니스 서비스 계층과 어플리케이션 서비스 계층과의 관계를 알게 되었다. RailCo와 TLS라는 두 회사에서 구축하는 SOA 시스템을 대상으로 계층에서의 연결 관계를 적용한다. 상향식으로 SOA을 접근한 첫 번째 회사 RailCo은 철도 부품 공급을 위해 설립되었다. 비효율적인 비즈니스 프로세스를 자동화하는데 구식 기술을 활용하고 있었고, 그로 인해 RailCo의 주요 고객인 TLS는 가격이 낮은 경쟁사들과 온라인 관계를 시작하게 되면서 RailCo는 뒤늦게 TLS만을 위한 웹 서비스를 서둘러 구축했다. RailCo는 오케스트레이션 서비스 대신 태스크 중심인 비즈니스 서비스가 어플리케이션 서비스들을 컨트롤할 수 있는 컨트롤러가 되어, 필요한 비즈니스 로직을 실행하는 컨트롤러로써 어플리케이션 서비스들을 조합하는 것이다. 하향식으로 SOA을 접근한 두 번째 회사 TLS는 분산 환경으로 구성된 기업으로 회계 요구사항이 증가되어, 인트라넷과 원격접속을 위한 정교한 웹 화면과 분석을 제공할 뿐만 아니라 전체 시스템의 여러 다른 모듈에 플러그인 할 수 있는 웹 서비스 어댑터를 제공한다. 그러나 회사의 인수와 그에 따른 프로세스 통합으로 회사의 정체성과 구조가 바뀌고 변화가 심한 비즈니스 모델을 처리해야만 했고 그에 맞는 SOA를 구축하려고 한다 [11].



(a) RailCo (b) TLS

(그림 7) 서비스 계층 구조 간의 연결

두 회사는 비즈니스로 서로 관계가 있고 위의 그림 7은 RailCo와 TLS 가 갖는 서비스의 계층 구조 이다. 그림7 (a)에서 RailCo는 오케스트레이션 계층에 서비스가 없음을 알 수 있다. 왜냐면 RailCo는 예산이 부족했고, SOA의 필요성을 알지 못했으며 TLS와 거래를 하기 위해서 B2B 흉내만 냈기 때문이다. 후에서야 SOA 시스템을 적용하려 했으나 TLS의 맞춤 서비스로 시스템을 개발하였기에 이상적인 SOA 시스템을 구축하기에는 어려움을 겪었다. RailCo에서 비즈니스 서비스와 어플리케이션 서비스의 연결 분석을 위하여 우선 그림 7(a)의 비즈니스 서비스들을 작업단위로 분할하였다. 그리고 난 후 어떤 작업과 어떤 오퍼레이션이 서로 연결

관계를 갖고 있는지를 표시하였다.

그림 7(b)의 TLS는 이미 서비스 지향 솔루션을 가지고 있었다. 그러나 회사의 인수와 그에 따른 프로세스 통합으로 인해 회사의 정체성과 구조가 바뀌고 변화가 심한 비즈니스 모델을 처리해야만 했고 그에 맞는 SOA를 구축하기 위해서 오케스트레이션 서비스 계층을 사용하기로 했다. TLS는 '*Time Sheet Submission*'을 위한 워크플로우 조직을 캡슐화하기 위해 WS-BPEL[21]로 프로세스를 정의했고, 모든 서비스의 액티비티를 컨트롤하며 성공적인 오케스트레이션 서비스 계층을 구현했다. 오케스트레이션 서비스 계층은 비즈니스 로직 추상화에 대한 확고한 기반을 확립하며, 확장된 어플리케이션 서비스 계층과 느슨한 연결 관계를 유지 할 것이다. 이 두 추상화 계층은 계속적으로 그 영역이 넓어질수록 향후 발생할 수 있는 요구사항 변경에 대응하는 시간이 감소할 것이다. 이 서비스들은 범용 적이고 재사용 가능하며 비즈니스에 독립적인 로직을 표현한다. 서비스 연결들은 작업과 오퍼레이션 사이에 연결 되어야 하며 위의 사례를 통한 결과가 아래 표 1에서 나타나 있다. 표 1(a) RailCo 서비스의 연결 테이블에 등장하는 여러 연결들 가운데 하나를 예로 들어보면 '*send an electronic invoice to Service*'라는 비즈니스 서비스의 하나의 작업은 '*export document to network folder*'라는 어플리케이션 서비스의 오퍼레이션과 메시지 전송을 통하여 연결된다는 것을 알 수 있다. 또한 어플리케이션 서비스에서 S_{A1} '*Legacy System*' 에는 '*export electronic invoice to network folder*', '*import electronic PO into accounting system*', '*send PO to accounting clerk's work queue*' 이렇게 3개의 서비스가 존재했지만, 앞서 말했듯 같은 계층에서 의존성이 강하여 하나의 서비스로 통합 하게 된다. 결과적으로 여러 종류의 문서를 처리하여 재 사용성을 높일 수 있는 오퍼레이션들이 남게 되어 표 1(a)에서 보듯, '*export document to network folder*', '*import and forward document to work queue*'가 존재함을 보였다. 표 1(b) TLS 서비스의 연결 테이블에 등장하는 여러 연결들 가운데 '*Reject Timesheet*'라는 비즈니스 서비스의 하나의 작업은 '*Comment*'라는 어플리케이션 서비스의 오퍼레이션과 메시지 전송을 통하여 연결된다는 것을 알 수 있다. 비즈니스 서비스에서 S_{B1} 'Invoice' 에는 '*get billed hours*', '*get billed period*' 이렇게 2개의 서비스가 존재했지만, 비즈니스 서비스에서도 역시나 같은 계층에서 의존성이 강하여 하나의 서비스로 통합하게 된다. 아래의 표 1의 분석 결과를 이용하여 서비스 연결 테스트의 테스트 요구사항 추출에 사용될 수 있으며, 이를 통하여 보다 연결 구조에 충실한 연결 테스트 데이터를 생성할 수 있는 기본 방안을 마련하였다.

(표 1) RailCo 와 TLS

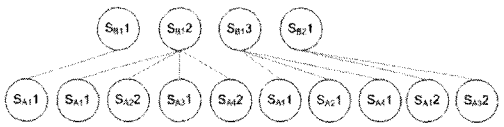
		S_{A1} -Legacy System		S_{A2} -Polling Notification		S_{A3} -Transform Accounting Documents		S_{A4}-Metadata Checking	
		S_{A1} -export document to network folder	S_{A1} -import and forward document to work queue	S_{A2} -poll folder for new documents	S_{A2} -if documents arrive for which there are subscribers, issue notifications	S_{A3} -transfor m XML docume nts to native format	S_{A3} -transfor m native docume nts to XML	S_{A4} -check if it is time to verify TLS metadata; if required, perform metadata check	S_{A4} -if metadata check fails, issue notification
S_{B1} -Invoice Submiss ion Service	S_{B1} -Issues messages compliant with the accounts payable WSDL	v							
	S_{B2} -request message	v			v	v			v
	S_{B3} -send an electronic invoice to Service	v		v				v	
S_{B2} -Order Fulfillm ent	S_{B1} -TLS Purchase Order Service		v				v		

(a) RailCo 서비스들의 연결 테이블

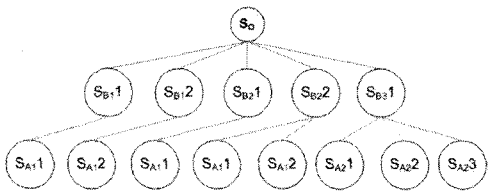
		S_{A1} -Human Resources Service		S_{A2} -Accounts payable Service			S_{A3} -Notification Service
		S_{A1} -Employee	S_{A2} -Comment	S_{A2} -Get Account	S_{A2} -Update Account	S_{A3} -Add Account	S_{A1} -Send Message
S_{B1} -Employee Service	S_{B1} -Update History	v					
	S_{B2} -Get Weekly Hours Limit		v				
S_{B2} -Timesheet Service	S_{B1} -Verify Timesheet	v					
	S_{B2} -Reject Timesheet	v	v				
S_{B3} -Invoice Service	S_{B1} - Get Billed Hours			v	v	v	

(b) TLS 서비스들의 연결 테이블

표 1로부터 아래 그림 8의 RailCo와 TLS의 SMFD를 표현한다.



(a) RailCo



(b)TLS

(그림 8) RailCo 와 TLS의 SMFD

그림 8에서 보여 진 SMFD에서 다음의 테스트 요구사항이 추출된다. RR(x)는 x을 수행한 경우의 결과를 의미하고, E(x)는 x에 대한 예상 결과를 의미한다. 정의 2에 의하여, RailCo의 테스트 요구사항 흐름은 다음과 같다.

TR={RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1),
RR(S_{B1}2,S_{A1}1)=E(S_{B1}2,S_{A1}1),
RR(S_{B1}2,S_{A2}2)=E(S_{B1}2,S_{A2}2),
RR(S_{B1}2,S_{A3}1)=E(S_{B1}2,S_{A3}1),
RR(S_{B1}2,S_{A4}2)=E(S_{B1}2,S_{A4}2),
RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1),
RR(S_{B1}3,S_{A2}1)=E(S_{B1}3,S_{A2}1),
RR(S_{B1}3,S_{A4}1)=E(S_{B1}3,S_{A4}1),
RR(S_{B2}1,S_{A1}2)=E(S_{B2}1,S_{A1}2),
RR(S_{B2}1,S_{A3}2)=E(S_{B2}1,S_{A3}2)}

RailCo의 테스트 요구사항은 SMFD에서 10개의 연결 관계를 가지고 있는데, 다시 말해 10개의 작업과 오퍼레이션의 연결 관계의 테스트를 받아야 하는 것을 의미한다.

예를 들어, RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1), S_{B1}3은 '*send an electronic invoice to Service*'라는 비즈니스 서비스의 하나의 작업은 '*export document to network folder*'라는 어플리케이션 서비스의 오퍼레이션과 메시지 전송을 통하여 연결된다는 것을 알 수 있다. 이 연결은 그림 8(a)을 보면 알 수 있는데 RR(S_{B1}3,S_{A1}1)=E(S_{B1}3,S_{A1}1)의 테스트 요구사항은 테스트 데이터가 S_{A1}1에서 S_{B1}3에게 기대하는 예상 출력 값을 확인하는 것을 의미 한다. S_{B1}3은 S_{A1}1에서 요구 메시지를 보냄으로써 이 트랜잭션을 시작한다.

TLS의 테스트 요구사항은 정의 2에 따라, 8개로 추출된다. S_O는 오케스트레이션 서비스 계층 미들웨어로서 서비스의 연결에 카운트 되지 않는다. 오케스트레이션 서비스는 처리를 제어하거나 관리를 한다. 그러므로 위의 그림 8(b)에 갖는 연결들의 작업과 오퍼레이션들은 오케스트레이션 서비스에게 제한된다. TLS 의 테스트 요구사항 흐름을 가질 수 있다.

TR={RR(S_{B1}1,S_{A1}1)=E(S_{B1}1,S_{A1}1),
RR(S_{B1}2,S_{A1}2)=E(S_{B1}2,S_{A1}2),
RR(S_{B2}1,S_{A1}1)=E(S_{B2}1,S_{A1}1), RR(S_{B2}2,S_{A1}1)=E(S_{B2}2,S_{A1}1),
RR(S_{B2}2,S_{A1}2)=E(S_{B2}2,S_{A1}2),
RR(S_{B3}1,S_{A2}1)=E(S_{B3}1,S_{A2}1),
RR(S_{B3}1,S_{A2}2)=E(S_{B3}1,S_{A2}2), RR(S_{B3}1,S_{A2}3)=E(S_{B3}1,S_{A2}3)}

예를 들어, RR(S_{B2}2,S_{A1}2)=E(S_{B2}2,S_{A1}2), S_{B2}2는 '*reject timesheet*'라는 비즈니스 서비스의 하나의 작업은 '*comments*'라는 어플리케이션 서비스의 오퍼레이션과 메시지 전송을 통하여 연결된다는 것을 알 수 있다. S_{B2}2는 S_{A1}2에서 요구 메시지를 보냄으로써 이 트랜잭션을 시작한다.

5. 결 론

서비스 지향 관점을 적용하면, 비즈니스 프로세스 모델이 변화에 대해 더 유연하고 기민하게 대응하는 구조를 가질 수 있다. 본 논문은 서비스들의 연결 특성을 기반으로 테스트 요구사항을 생성하기 위해 SOA 특성을 표현한 SMFD를 정의하였다. SOA 계층구조 사이의 연결 관계의 복잡도를 중심

으로 서비스 연결 테스트 방안을 제안하였고, 이를 제안한 방법으로 서비스 지향 아키텍처를 갖는 회사의 서비스 연결에 적용하였다. 현재 SOA 시스템이 실제 프로젝트로 연결되지 않는 주요 요인 중 하나로 서비스 연결 테스트 부재[4]의 문제와 SOA에서의 서비스 통합과 서비스 연결에 대한 차이를 느끼지 못한다는 문제로 그에 따른 서비스 계층 간의 연결에 대한 테스트의 필요성이다. 이는 기존의 서비스 통합 테스트와는 다른 관점의 서비스 연결 구조를 SOA에서 요구하기 때문이다.

본 논문에서는 SOA에서의 서비스 연결 테스트를 위하여 계층 간의 특성을 분석하여 같은 계층 간의 연결이 아니라 서로 다른 계층에서 연결이 이루어져야 하며, 같은 계층에서의 연결은 서비스끼리의 의존성이 강한 경우 서비스의 통합임을 보이고, 서비스 통합과 서비스 연결의 차이점을 분석하였다. 이를 표현하기 위하여 SMFD을 정의하였고, 또한 SMFD을 구성하기 위해 연결 테이블을 작성하여, 서비스의 연결 관계를 비즈니스 서비스의 작업과 어플리케이션 서비스의 오퍼레이션의 연결로 분석하고, 이 연결 관계를 테스트에서 반드시 처리해야 하는 요구사항을 추출하였다.

본 연구는 SOA 개발자의 대부분은 웹서비스 개발자들이라는 현재 상황을 고려하여, 본 논문에서 제안한 테스트 설계를 기존의 웹서비스 테스트 도구에 적용하는 연구도 앞서 진행하였다[22]. 기존 웹 서비스 개발자들을 지원하는 SOA 테스트 방안 개발 노력의 일환으로서, 현재 웹 서비스 테스트 도구들의 특징 파악을 하고 웹 서비스 테스트 도구 중심으로 SOA 연결 테스트의 구체적인 사례연구를 수행하고 있다. 동시에 본 논문의 테스트 요구사항을 만족시키는 테스트 데이터 선정 기준 개발을 진행하고 있다.

참 고 문 헌

[1] M.P Papazoglou, D. Georgakopoulos, "Service-Oriented Computing", Communication of ACM, Vol.46 No.10, pp25-28, 2003.10

[2] V.Kapoor, "Service and Automatic Computing: A Practical Approach for Designing Manageability", in proceeding of the 2005 IEEE International Conference on Service Computing(SCC'05), Vol.2, pp41-48, July 2005

[3] Eric A. Mark. A Planning and Implementation Guide for Business and Technology, John Wiley and Sons, 2006.

[4] "HP Software Universe 2007", http://h30350.www3.hp.com/conference/index.jsp

[5] W3C Web Service Activity, "http://www.w3org/2002/ws"

[6] David Sprottand Lawrence Wilkes, "Understanding SOA", CBDI Journal 2003,9

[7] F.Curbera,M.Duftler, R. Khalaf, W.Nagy, N.Mukhi, and S. Weerawarana. "Unraveling the Web Services Web: An Introduction to SOAP,WSDL, and UDDI."IEEE Internet Computing, March 2002

[8] 신민철, "기초에서 실무까지 XML 웹 서비스" 프리첼 April.2004

[9] 이상민, "비즈니스 신속성, SOA가 책임진다." Oracle Korea Magazine,(Summer 2005), p.34.

[10] Thomas Erl, Service-Oriented Architecture - A Field Guide to integrating XML and Web service, Prentice Hall, 2004

[11] Thomas Erl, Service-Oriented Architecture - Concepts, Prentice Hall, 2005

[12] SoapUI - Web Service Testing, http://www.soapui.org/

[13] JUnit - Test Driven Development, http://www.junit.org

[14] PushToTest TESTMAKER, http://www.pushtotest.com

[15] webInject - web/HTTP Test Tool, http://www.webinject.org

[16] Paul Ammann and Jeff Offutt, Introduction to Software Testing, Cambridge, 2008

[17] Hoijin Yoon, Eun Mi ji, and Byoungju Choi, "Generating Test Requirements for the Service Connections based on the Layers of SOA" in proceeding on ICST, April 2008.

[18] Jeff Offutt and Wuzhi Xu, "Generating Test Cases for Web Services Using Data Perturbation," in Workshop on Testing, Analysis and Verification of Web Services, pp.41-50, July 2004

[19] Robert Peterson, "Get started with WebSphere Integration Developer," IBM WebSphere Developer Technical Journal, Dec. 2005

[20] Daniel A. Menascé, "Composing Web Services: A QoS View," IEEE Internet Computing, pp.88-90, Nov./Dec. 2004

[21] Dieter Konig, "Web Services Business Process Execution Language(WS-BPEL)", OASIS Open Standards Day XTech 2005 Conference Amsterdam, 2005.

[22] Hoijin Yoon, Eun Mi ji, and Byoungju Choi, "Building Test Steps for SOA Service Orchestration in Web Service Testing Tools" in proceeding on ICUIMC, Jan. 2008.

◐ 저 자 소 개 ◑

지 은 미(Eun Mi Ji)
2005년 상명대학교 소프트웨어공학과 졸업(학사)
2008년 이화여자대학교 대학원 컴퓨터학과 졸업(석사)
2008년 ~ 현재 티맥스코어 연구원
관심분야 : 소프트웨어공학, 서비스기반아키텍처, 테스팅
E-mail : jjiem@naver.com

윤 회 진(Hoijin Yoon)
1993년 이화여자대학교 전자계산학과 졸업(학사)
1998년 이화여자대학교 대학원 컴퓨터학과 졸업(석사)
2004년 이화여자대학교 대학원 컴퓨터학과 졸업(박사)
2007년~현재 협성대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 소프트웨어테스트, 서비스기반아키텍처, 컴포넌트소프트웨어테스트
E-mail : hjyoon@uhs.ac.kr

최 병 주(Byoungju Choi)
1983년 이화여자대학교 수학과 졸업(학사)
1988년 Purdue University 전산학과 졸업(석사)
1990년 Purdue University 전산학과 졸업(박사)
1995~현재 이화여자대학교 컴퓨터학과 교수
관심분야 : 소프트웨어공학, 소프트웨어테스트, 임베디드시스템테스팅, 서비스기반아키텍처
E-mail : bjchoi@ewha.ac.kr