코레어그래피 기반 웹 서비스 조합의 구현 적합성 평가 및 테스트 방법☆

Conformity Assessment and Test Method for Implementation of Web Services Compositions based on Choreography

국 승 학* 서 용 진** 김 현 수***
Seung-hak Kuk Yong Jin Seo Hyeon Soo Kim

요 약

Recently Web services and SOA(Service Oriented Architecture) is widely used for an effective integration way of distributed applications. The service choreography is a way of service compositions to define workflows and the message exchange protocols among several partner ser-vices in the B2B business process environment. It defines the overall business process from a global perspective. However, it must be transformed into some kind of the executable form to implement service compositions, because it does not support an execution mechanism but is a de-clarative form. Therefore, the transformed executable model must be validated with the original choreography model to guarantee behavior equivalence. In this paper, we propose the conformity assessment with respect to service compositions method between the choreography model and the transformed executable model. And we suggest the test method to show behavior equivalence between them from a viewpoint of the dynamic execution.

ABSTRACT

최근 웹 서비스 기반 서비스 지향 구조가 분산된 애플리케이션의 효율적인 통합을 위한 방법으로 널리 활용되고 있다. 서비스 지향 구조에서의 코레어그래피는 기업 간 비즈니스 프로세스 환경에서 메시지 교환에 초점을 맞춘 협업 방식으로 참여하는 서비스들 사이의 작업 진행 순서와 메시지 교환 프로토콜을 정의함으로써 거시적인 측면에서 전체 프로세스를 모델링하기 위해 사용된다. 그러나 이러한 코레어그래피는 웹 서비스 조합을 실행 시키는 방법이 아니기 때문에 구현 모델로 변환 후 사용된다. 따라서 코레어그래피 모델을 실행시키기 위한 구현 모델은 기능적인 측면에서 코레어그래피 모델을 정확하게 구현하였는지 반드시 평가되어야 한다. 이에 본 논문에서는 코레어그래피 모델과 구현 모델의 서비스 조합 적합성을 평가하는 방법과 수행을 통한 동작의 일치성을 보장하기 위한 테스트 방법을 제시한다.

☞ keyword : Service Oriented Architecture(서비스 지향 구조), Web Services Composition(웹 서비스 조합), Choreography(코레어그래피), Software Test(소프트웨어 테스트)

1. 서 론

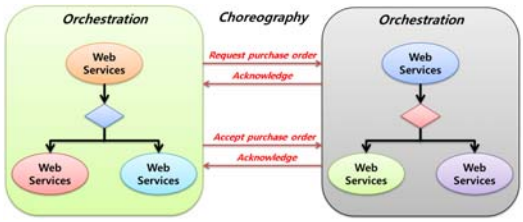
최근 비즈니스 환경에서는 능동적으로 대처 가능한 IT 환경이 요구되고 있으며 이를 위해 웹 서비스 기반 SOA(Service Oriented Architecture: 서비스 지향 구조)가 널리 쓰이고 있다. 이러한 SOA는 잘 정의된 인터페이스와 서비스들 간 계약을 통해, 서비스로 정의되는 애플리케이션의 다양한 기능 단위를 상호 연관시키는 컴포넌트 모델로 지속적인 업무 처리 변화를 시스템에 빠르게 반영하고자 하는 수요를 대응하기 위한 방법이다[1]. 오늘날의 서비스는 일반적으로 XML 기반의 웹 서비스(Web Services)로 구현되고 있다. 웹 서비스 기술은 서비스 명세를 위한 WSDL(web services definition language), 메시지 교환을 위한 SOAP(simple object access protocol), 서비스 등록 및 검색을 위한 UDDI(universal description, discovery, and integration)등의 기술을 이용해 SOA를 가장 잘 구현하고 있다[2].

그러나 SOA는 웹 서비스 기술만으로 완성되는 것은 아니다. 웹 서비스는 하나의 단위 서비스를 구현하기 위

* 정 회 원 : 충남대학교 컴퓨터공학과(공학박사)
triple888@cnu.ac.kr
** 정 회 원 : 충남대학교 컴퓨터공학과 석사재학
yjseo082@cnu.ac.kr
*** 정 회 원 : 충남대학교 컴퓨터공학과 교수
hskim401@cnu.ac.kr(교신저자)
[2011/10/24 투고 - 2011/10/27 심사 - 2011/11/23 심사완료]
☆ 이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업 연구임(KRF-2008-521-D00400).

한 방법일 뿐 서비스를 효율적으로 조합하기 위한 기술은 아니다. 서비스 조합을 통해 새로운 애플리케이션을 개발하는 경우 전체 애플리케이션이 서비스들 간 워크플로우를 어떻게 수행하는지에 대한 정의가 필요하다. 현재 SOA에서 서비스의 조합을 기술하기 위한 대표적인 두 가지 방법은 오케스트레이션(Orchestration)과 코레어그래피(Choreography)이다[3]. 오케스트레이션은 기업의 내부적인 비즈니스 프로세스에 주로 사용되며, 중앙 프로세스가 관련 서비스들을 통제하고 각각의 서비스가 실행하는 작업을 조율하는 방식으로 서비스 통합을 구현한다. 반면에 코레어그래피는 기업 간 비즈니스 프로세스 환경에서의 메시지 교환에 초점을 맞춘 협업 방식이며, 중앙의 코디네이터에 의존하지 않는다. 이는 참여하는 서비스들 사이의 작업 진행 순서와 메시지 교환 프로토콜 등을 정의한 것으로, 참여하는 다양한 서비스들 사이의 계약된 행위를 보장하기 위해 사용한다. 이러한 코레어그래피는 일반적으로 다수의 기업이 참여하는 웹 서비스 조합에서 각 참여자의 역할과 조합된 서비스들 사이의 상호작용을 명세하기 위해 사용한다.

그러나 코레어그래피는 오케스트레이션과 달리 실질적인 서비스 조합을 실행하지 않는다. 따라서 코레어그래피로 정의된 서비스 조합을 실행하기 위해서는 실행 가능한 형태의 서비스 조합에 대한 구현이 필요하다. 이런 이유로 인해 코레어그래피로 정의된 비즈니스 프로세스와 실행 가능한 구현 모델 사이에서 동작의 일치성을 보장하기 위해 적합성 검증 및 동적인 실행 측면에서의 기능 테스트가 수행되어야 한다. 이에 본 논문에서는 코레어그래피 기반 웹 서비스 조합을 테스트하기 위한 방법을 제시한다. 이를 위해 코레어그래피 모델과 구현된 웹 서비스 조합의 적합성을 평가하는 방법을 제시하고 구현된 웹 서비스 조합의 수행을 통한 동작의 일치성을 보장하기 위한 테스트 방법을 제시한다. 우선 코레어그래피 서비스 조합 모델과 구현된 서비스 조합이 일치하는지에 대한 검증은 코레어그래피 모델과 구현된 모델을 대상으로 모델 요소의 분석을 통해 적합성을 검증하는 작업을 수행한다. 또한 코레어그래피 모델과 구현된 모델 사이의 적합성이 검증되었다 하더라도 그 기능이 오류가 없음을 보장할 수 없다. 따라서 실제 구현된 서비스 조합에 대한 동적인 테스트를 통해 그 기능이 정확히 구현되었는지 판단할 필요가 있다. 이에 본 논문에서는 구현된 서비스 조합을 테스트하기 위해 테스트 케이스의 생성, 테스트 드라이버의 생성 등과 같이 동적인 테스트 방법을 제시한다.



(그림 1) 오케스트레이션과 코레어그래피의 관계

2. 관련 연구

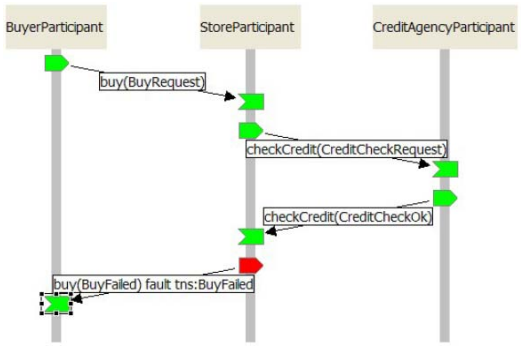
2.1 웹 서비스 조합

웹 서비스는 다른 웹 서비스를 호출하지 않고 자신의 서비스 실행 결과만 반환 하는 단일 서비스(atomic service)와 하나의 서비스 내에서 프로세스 실행 흐름에 따라 외부의 다른 서비스 호출을 포함하는 조합 서비스(composite service)로 구분할 수 있다. 웹 서비스 조합이란 복합 서비스를 개발하는 일련의 과정이라 정의할 수 있다. 웹 서비스 조합(composition)에서의 코리어그래피(choreography)와 오케스트레이션(orchestration)은 웹 서비스 참여자의 외부 웹 서비스들의 순차적인 프로세스를 구성하고, 내부 비즈니스 논리대로 웹 서비스를 실행시키는 핵심 요소이다. (그림 1)은 두 웹 서비스 파트너들이 웹 서비스로 통신 시 코레어그래피와 오케스트레이션 부분을 표시하고 있다[3].

코레어그래피는 여러 파트너 사이에서 일어나는 메시지 교환 방식을 기반으로 일반적으로 둘 이상의 웹 서비스 참여자가 상호작용을 할 때 생길 수 있는 웹 서비스의 프로세스로 정의된다. 이러한 코레어그래피는 일반적으로 오케스트레이션에 비해 복잡한 웹 서비스 조합에 사용된다. 다수의 참여자들의 역할과 그들 사이의 상호작용을 명세함으로써 거시적 관점에서의 서비스 조합을 명세하기에 적합한 방법이다. 현재 일반적으로 W3C의 WS-CDL(Web Services Choreography Description Language)을 이용해 표현한다[4]. WS-CDL은 코레어그래피에 참여한 웹 서비스의 정보를 담고 있으며 비즈니스 워크플로우 전체를 기술하는데 용이한 언어이다. WS-CDL은 아래 (그림 2)와 같은 구조를 가지며 크게 데이터 타입의 정의 부분과 코레어그래피 구성 부분으로 나눌 수 있다. 조합에 참여한 참여자와 그 연관자의 정보는 타입 정의 부분 중 <roleType>, <relationshipType>, <participantType>, <channelType>에서 정의되며, 메시지와 관련된 정보는

```
<package name="NCName" author="xsd:string"?
    version="xsd:string"? targetNamespace="uri"
    xmlns="http://www.w3.org/2005/10/cdl">
    <informationType/>*
    <token/>*
    <tokenLocator/>*
    <roleType/>*
    <relationshipType/>*
    <participantType/>*
    <channelType/>*
    Choreography-Notation *
</package>
```

(그림 2) WS-CDL 템플릿



(그림 3) pi4Soa에서의 코레어그래피 테스트

<informationType>, <token>, <tokenLocator> 등으로 정의된다.

반면 오케스트레이션은 한 파트너에 대하여 내부 비즈니스 프로세스 논리를 정의하기 위해 사용하며, 현재 OASIS(Organization for the Advancement of Structured Information Standards)에서 정의한 BPEL이 가장 많이 사용되고 있다[5]. BPEL 프로세스는 프로세스에 참여하는 웹 서비스들의 정확한 호출 순서(순차적, 병렬적)와 조건부 실행문에 대한 정의를 지원한다. 이를 통해 루프 정의, 변수 선언, 변수 값 할당/복제, 오류 처리 등을 정의할 수 있으며 이런 요소들을 조합하여 알고리즘에 기반한 복잡한 비즈니스 프로세스를 정의할 수 있다. BPEL 비즈니스 프로세스는 요청을 처리하기 위해 관련된 웹 서비스들을 호출하고 그 결과를 반환하는 방식으로 동작한다. BPEL 프로세스가 다른 웹 서비스와 커뮤니케이션을 수행하는 과정에서는 각 웹 서비스에 대한 WSDL 기술이 매우 중요하게 활용된다.

2.2 웹 서비스 조합 테스트에 관한 기존 연구

2.2.1 pi4soa 프로젝트

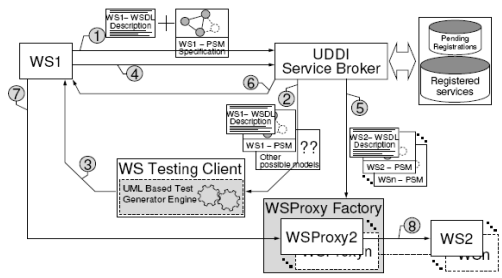
pi4soa 프로젝트는 상호작용에 대한 SOA 기반 분산 시스템의 작성을 돕는 도구를 제공한다[6]. pi4soa에서 제공하는 여러 기능 중 코레어그래피 검증 프레임워크와 서비스 검증 프레임워크는 기술한 WS-CDL과 웹 서비스의 검증 기능을 통해 상호작용에 대한 테스트를 수행할 수 있다. 도구를 통해 WS-CDL을 작성하고 가시적으로 시나리오를 작성 후 시뮬레이션을 통해 검증하게 되는데, 이때 작성한 WS-CDL의 요소와 속성을 분석하여 시뮬레이션 하게 된다. 본 연구에서와 동일한 방식으로 기술 문서(Description Document)를 분석한다.

하지만 이 도구에서 코레어그래피 및 서비스 검증 프레임워크는 어디까지나 실행 이전의 WS-CDL을 기반으로 분석하고 시뮬레이션 하므로 실제 수행시의 상호작용 검증과 동일하다고 볼 수 없다. WS-CDL에 따라 정확히 구현했는지를 알기 위해서는 실행 시점에서의 상호작용을 검증할 필요가 있다. 따라서 본 연구에서는 pi4soa에서는 다루지 않았던 실행 시의 실제 상호작용 여부를 확인하는 기법을 제안한다.

2.2.2 UDDI를 이용한 웹서비스 조합 상호운용성 테스트 프레임워크

논문 [7]에서는 코레어그래피 기반의 서비스 조합의 상호 운용성을 테스트하기 위한 확장된 UDDI 레지스트리를 제안하였다. 확장된 UDDI는 코레어그래피에 참여하는 개별 서비스들을 UDDI에 등록할 때 새로운 서비스가 조합에 적합한지 검증하는 역할을 수행한다. 논문 [7]에서 제안한 프레임워크는 Audition이라고 명명되었다.

(그림 4)는 이 연구에서 제안한 Audition 프레임워크를 나타낸다. 그림의 ①~⑧이 설명하는 바는 다음과 같다.① 웹서비스 WS1은 호출(invocation)을 받기 위한 서비스로 출판되기 위해 UDDI 레지스트리를 요청한다. WS1은 제공된 서비스의 문법을 정의해 놓은 WSDL 파일, PSM (Protocol State Machine)의 XMI 파일 생성 등을 제공한다. 만약 이미 기술(description)이 등록되어 있다면 웹서비스 접근 포인트(access point)가 제공된다. ② UDDI 서비스는 WS1을 연관된 데이터베이스에 넣는데 이때의 데이터베이스는 실제 레지스트리가 아닌 임시 저장 공간이다. 성



(그림 4) Audition Framework

공적으로 이 단계가 수행되면 웹서비스 테스터(WS Tester)를 생성한다. 웹서비스 테스팅 클라이언트(WS Testing Client)는 UML 기반 테스트 생성 엔진에 의해 만들어지며 이 클라이언트는 재사용되는 것을 목표로 한다. ③ WS 테스팅 클라이언트가 시작되면 WS1은 다른 서비스들을 호출하기 시작하고 테스트 세션의 드라이버 역할을 하게 된다. 이 연구의 주된 목적은 2개의 서로 다른 서비스 간의 상호작용에서 발생하는 문제점을 파악하는 것에 있다. ④ audition 프레임워크의 수행 동안, 기본적인 서비스가 아니라면 - 예를 들어 다른 서비스의 협력을 요구하지 않는 서비스 - WS1은 서비스 수행 준비의 완료를 위해 필요한 다른 서비스의 참조를 위해 UDDI 서비스를 요청한다. ⑤ 임시 상태(pending state)에 있는 서비스를 참조하고자 하는 경우 UDDI를 체크한다. 서비스가 임시 상태에 있는 경우 UDDI는 WS factory를 사용해서 요청된 서비스의 WS Proxy를 생성한다. 이 WS proxy는 요청된 서비스와 같은 인터페이스를 구현하고 그 결과는 웹서비스를 시뮬레이션하기 위한 PSM의 입력값으로 쓰인다. ⑥ UDDI 서비스에서 WS1에 의해 생성된 각각의 조회 요청은 요청된 서비스의 프록시 버전에 대한 바인딩 참조를 반환한다. ⑦ UDDI 서비스에 의해 제공된 참조를 바탕으로 WS1은 요청된 서비스의 프록시 버전에 대한 호출을 생성한다. 프록시 버전에서는 서비스의 내용과 WS1에 의해 만들어진 호출들의 순서를 체크할 수 있다. 특히 이 연구에서는 클라이언트 서비스에 의해 만들어진 호출의 순서를 고려하였다. 에러가 발생하면 현재 프록시에 의해 모니터링 되고 있는 부분을 확인하고 UDDI 서비스에 알린다. 그 결과로써, 주소록 서비스(directory service)는 현재 테스트 중인 서비스를 임시 엔트리(pending entry)로부터 지우고 등록을 취소한다. ⑧ 호출이 발생하면 프록시 서비스가 실제 구현된 서비스를 호출하고 호출했던 서비스(WS1)에 결과값을 반환한다. 이전 테스팅 클라이언트에 의해 만들어졌던 호출에 기반해서 프로세스를 계속 진행한다.

이 프레임워크에서의 상호작용 테스트는 먼저 UDDI에서 임시 상태(pending state)를 기록할 임시 저장 공간(pending registry)이 필요하며 서비스 별로 웹서비스 프록시를 만들어야 하기 때문에 웹서비스 호출이 늘어나면 오버헤드가 커질 수 있다는 단점이 있다.

2.2.3 웹 서비스 조합 테스트

그 외의 많은 연구들 또한 웹 서비스 조합에 대한 테스트 방법을 제안하고 있다. [8]의 경우 오케스트레이션 기반의 웹 서비스 조합의 화이트 박스(white-box) 단위 테스트를 위한 프레임워크를 제안하였다. 이 논문에서 제안하는 프레임워크는 비즈니스 프로세스를 하나의 조합된 서비스로 보고 해당 프로세스의 실행 흐름에 따라 테스트를 수행한다. [9]도 오케스트레이션에서의 단위 테스트를 위한 화이트 박스 테스트 방법을 제안하였다. [8]에서는 테스트 케이스 생성에 관한 점을 고려하지 않은 반면 [9]에서는 제어 흐름 커버리지 (control flow coverage) 기준 적용을 통한 테스트 케이스 생성 방법에 대해 제안하였으며, 제어 및 데이터 흐름을 통한 모델 체킹(model checking)도구를 제안하였다. [10]에서는 OWL-S를 새로운 프로세스로 자동으로 변환하는 통합 프로세스를 제안하였다. 이 과정에서 기존의 모델 체킹 도구인 BLAST를 이용하여 정의된 비즈니스 프로세스를 검증하는 방법을 제시하였다.

기존의 테스트 노력들은 서비스 조합에 있어서 단위 및 통합 테스트를 위한 테스트 케이스 생성, 모델 체킹 등을 통한 프로세스의 검증, 그리고 서비스 조합의 적합성 테스트에 대해 초점을 맞추고 있다. 그러나 코레어그래피 기반 서비스 조합에 있어 구현 모델의 적합성 검증에 관한 연구는 수행되지 않았다. 또한 코레어그래피 모델이 구현 모델에서 정확히 동작하는지 테스트하는 방법 또한 연구되지 않았다. 그러나 본 논문에서는 코레어그래피 기반 웹 서비스 조합에서 구현 모델과 코레어그래피 모델의 참여서비스, 데이터 타입 및 메시지 포맷, 메시지 교환 프로토콜, 워크플로우의 비교를 통해 구현 적합성을 평가한다. 또한 코레어그래피 모델의 워크플로우 분석을 통해 구현 모델의 테스트를 위한 테스트 케이스를 생성하고 이를 기반으로 테스트 수행하기 위한 방법을 제시한다.

3. 코레어그래피 기반 서비스 조합 테스트 방법

코레어그래피 기반 서비스 조합을 테스트하기 위해서는 우선 코레어그래피 모델과 구현 모델의 정의가 필요하다. 앞서 설명한 것과 같이 코레어그래피 모델은 거시적 관점에서의 서비스 조합을 기술하기 위한 방법으로 전체 비즈니스 프로세스를 기술하기 위해 사용된다. 따라서 코레어그래피 모델을 통해 서비스들간의 협력 관계, 프로토콜 및 교환 메시지에 대해 정의한다. 그러나 이러한 코레어그래피 모델은 실질적인 서비스 조합을 수행하는 모델이 아니기 때문에 실제 구현은 WS-BPEL과 같은 실제 수행 가능한 형태로 구현되어야 한다. 본 논문에서는 코레어그래피 모델에 대한 WS-BPEL 모델을 구현 모델로 정의한다. 따라서 코레어그래피 기반 서비스 조합의 테스트는 두 가지 관점에서 수행되어야 한다.

- 코레어그래피 모델과 구현 모델의 적합성 테스트: 구현 모델이 코레어그래피 모델에 따라 잘 구현되었는지 검증
- 구현 모델 기능 테스트: 구현 모델에 대한 동적인 실행 측면에서의 기능 테스트

예를 들어, 다양한 회사들 간의 비즈니스 업무를 하나의 조합 서비스로 개발하는 경우 각 회사의 서비스의 조합 순서 즉 워크플로우, 주고 받는 메시지 포맷, 메시지 교환 프로토콜에 관한 내용이 코레어그래피로 정의된다. 따라서 실제 구현된 서비스 조합이 이러한 모델에 따라 구현되었는지 검증하는 것이 필요하다. 동적인 테스트는 코레어그래피로 정의된 서비스 조합이 실제 동작 과정에서 문제없이 동작하는지 검증하는 것이다. 실제 구현된 서비스 조합이 코레어그래피 모델과 적합하도록 개발되어 있더라도 그 모델 자체에 오류가 있다면 적합성의 검증 여부가 무의미하다. 따라서 정의된 서비스 조합이 오류가 없음을 검증하는 작업이 수행되어야 한다.

3.1 코레어그래피 기반 서비스 조합 예제

본 논문에서 코레어그래피 기반 서비스 조합 테스트 방법을 설명하기 위해 다음과 같은 예제를 사용한다.

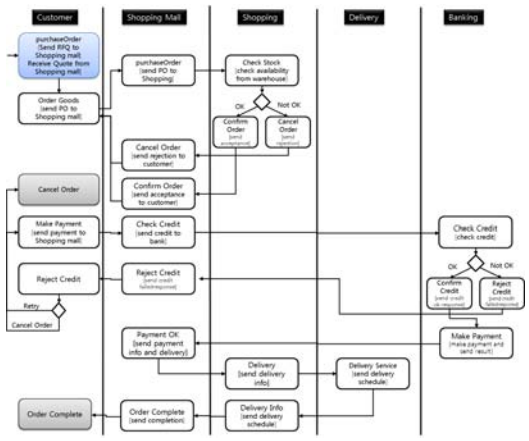
(그림 5)의 예제는 물품 판매자 역할을 수행하는 다수의 Shopping Service와 배송을 위한 Delivery Services, 결제를 위한 Bank Service와 다수의 Shopping Service에 대한 포탈 서비스를 제공하는 Shopping Mall Por-tal Service

(그림 5) 웹 서비스 조합 예제

로 구성된다. 위의 예제는 다음과 같이 작업을 수행한다.

① 고객의 조회요청을 처리하기 위해 Shopping Mall Portal Service는 해당 물품에 대한 목록을 Shopping Services의 서비스를 이용해 수집한다.
② 고객의 구매요청을 위해 Shopping Mall Portal Service는 해당 Shopping Service에 구매를 요청한다.
③ 고객의 물품대금 결제 요청에 대해 Shopping Mall Portal Service는 Bank Service를 통해 해당 결제 가능 여부를 확인한 후 Shopping Service에 실질적인 결제를 요청한다. Shopping Service는 Bank Service를 통해 해당 물품에 대한 결제를 수행한다. 결제가 완료되면, Shopping Service는 Delivery Service를 통해 해당 물품을 배송한다.
④ 고객 자신이 구매했던 기존의 물품에 대한 목록을 조회할 수 있다. 이를 Shopping Mall Portal Service에 요청하면, Shopping Mall Portal Service는 각각의 Shopping Service에 해당 사용자의 구매 이력을 요청하고, 취합된 정보를 사용자에게 보여준다.
⑤ 또한 각 Shopping Service와 Shopping Mall Portal Service의 결제 이력을 확인할 수 있는 기능을 제공한다. 각각의 Shopping Service는 Shopping Mall Portal Service와 Banking Service를 통해 해당 Shopping Service에서 발생한 결제 및 금전 거래에 대한 이력을 확인한다.

예제의 서비스는 크게 물품 조회, 구매, 결제, 배송, 결제이력 조회라는 다섯 가지 기능을 다양한 Shopping Service, Delivery Service, Shopping Mall Portal Service, Bank Service의 조합을 통해 제공한다. 이러한 기능은 코레어그래피를 통해 조합을 표현하며 각각의 참여 서비스들 간 역할 및 동작, 메시지 교환 방법을 기술한다. 예를



(그림 6) 코레어그래피 예제

들어 제품 구매 및 결제에 대한 서비스 조합을 코레어그래피로 표현할 경우 다음과 같이 나타낼 수 있다. (그림 6)은 메시지의 전달 순서에 따른 제품의 구매 및 결제 과정의 전체 흐름을 보여 준다.

3.2 코레어그래피 기반 서비스 조합 구현 적합성 테스트

앞서 설명한 것과 같이 코레어그래피 기반 서비스 조합 모델은 결과적으로 동작 가능한 형태의 서비스 조합으로 구현해야 한다. 따라서 코레어그래피 모델과 실제 구현된 모델 사이에 적합성을 테스트하여 구현된 결과물이 조합 의도에 적합한지 검증되어야 한다. 이러한 적합성을 검증하기 위해서는 코레어그래피 모델과 구현 모델이 일치하는지를 확인한다. 그러나 코레어그래피 기반 서비스 조합을 기술하는 방법과 이를 구현하는 방법이 다양하기 때문에 모든 상황을 표현하기 어렵다. 따라서 본 논문에서는 코레어그래피 모델을 정의하기 위해 가장 많이 사용되는 언어인 WS-CDL과 이를 구현하기 위한 WS-BPEL을 기반으로 적합성 검증 방법을 설명한다.

코레어그래피 모델과 구현 모델 사이의 적합성 검증을 위해서는 다음과 같은 네 가지 측면에서의 검증이 필요하다.

1) 데이터 타입 및 메시지 포맷의 일치 여부

실제 구현된 서비스 조합이 코레어그래피를 기반으로 정확히 구현되었는지 확인하기 위해서는 정의된 조합과 구현된 조합이 사용하는 데이터 타입 및 메시지 포맷이 일치하는지 확인해야 한다. 이를 위해서는 코레어그래피에서 정의한 메시지 포맷과 구현된 모델에서 사용하는 메시지 포맷을 비교하여 그 일치 여부를 확인한다. WS-CDL의 경우 데이터 및 메시지 포맷을 정의하기 위해 Information Types, Variables, Expressions, 그리고 Tokens가 사용된다. 각각에 대한 설명은 다음과 같다.

- Information Types: XML Schema, WSDL문서에 정의된 데이터 타입을 이용해 코레어그래피 내에서 사용할 데이터 타입을 정의한다.
- Variables: 코레어그래피 내에서 사용할 데이터 구조를 정의하기 위해 사용된다. Information Types에서 정의된 데이터 타입을 이용하여 실제 송/수신 메시지 혹은 내부에서 사용하게 될 데이터 구조를 정의한다.
- Expressions: WS-CDL 내부의 특정 데이터를 참조, 변경하거나 기존의 데이터로부터 새로운 데이터 타입을 생성하기 위해 사용된다. 일반적으로 사용하는 표현 방법과 리터럴을 이용해 새로운 변수(variables)를 생성하는 것이 가능하며, WS-CDL 내의 특정 정보를 질의 표현(Query Expression)을 이용해 참조하는 것이 가능하다.
- Tokens: 이는 변수 혹은 메시지의 일부 데이터를 참조할 때 사용되는 별칭(alias)을 나타내고자 할 때 사용된다. 메시지 내에서 일부 데이터를 추출하거나 참조하기 위해 사용되며 tokenLocator를 이용해 실제 메시지 내의 참조할 데이터의 위치를 찾아낸다.

위와 같이 코레어그래피 기반 서비스 조합에서 사용하는 데이터 및 메시지의 포맷은 이를 구현하는 시스템에서 대응되는 요소로 표현되어야 한다. WS-BPEL의 경우 데이터 타입은 기존의 WSDL 및 XML Schema에서 정의한 것을 사용한다. 송/수신 메시지 및 내부의 데이터 구조는 <variables>를 이용하여 정의한다. 또한 <property>와 <propertyAlias>를 이용해 코레어그래피의 Token, TokenLocator를 표현할 수 있다.

WS-CDL의 경우 Information Types, Variables, Expressions, Tokens을 이용해 데이터 타입과 메시지 포맷을 정의한다. 그러나 이 중에서 실질적인 데이터 타입 및 메시지 포맷을 정의하는 부분은 Information Types, Variables의 두 가지 요소이다. 따라서 WS-CDL 내에 정의된 Information Types, Variables에 대한 XML Schema 정의를 추출한다. Tokens의 경우 별칭을 기술하기 위해 사용한다. 같은 데

```
<participantType name="ShoppingService">
    <roleType typeRef="tns: ShoppingServiceRole"/>
</participantType>
……
<roleType name=" ShoppingServiceRole">
    <behavior name=" ShoppingServiceBehavior"
    interface= "purchaseOrder"/>
</roleType>
```

(그림 7) WS-CDL에서 서비스 참여자 정의

이터 타입에 대한 이름이 달라질 수 있기 때문에 별칭 정보를 함께 저장해야 한다. 구현 모델인 WS-BPEL에서는 Variables 요소를 통해 데이터 타입 및 메시지 포맷을 기술한다. WS-BPEL 역시 이러한 정보에 대한 구조를 XML Schema를 이용해 정의하기 때문에 각 Variables 요소에 대한 XML Schema 정의를 도출해야 한다. 그러나 코레어그래피 모델은 여러 구현 모델로 나누어져 구현될 수 있기 때문에 참여하는 모든 서비스에 대한 구현 모델을 참조하여 검증해야 한다. 위와 같이 데이터 타입 및 메시지 포맷의 일치 여부를 판단하기 위해서 WS-CDL과 WS-BPEL에 기술된 데이터 타입 및 메시지 포맷에 대한 비교를 수행해야 하며, 각각의 일치 여부는 다음과 같이 요약할 수 있다.

2) 참여 서비스의 일치 여부

코레어그래피 모델에서는 서비스 조합에 참여하는 서비스를 참여자(participant)로 정의하며 실제 서비스 인터페이스는 각 참여자의 역할(roleType)을 통해 정의한다.

```
<partnerLinkType name="ShoppingMallShoppingLink">
    <role name=" ShoppingServiceRole "
    portType="purchaseOrderPT" />
    <role name=" ShoppingMallServiceRole "
    portType="purchaseOrderPT" />
</partnerLinkType>
……
<portType name="purchaseOrderPT">
      <operation name="purchaseOrder">
      ……
      </operation>
</portType>
```

(그림 8) WS-BPEL에서 서비스 참여자 정의

이는 아래의 WS-CDL 예제처럼 실제로 구현될 서비스의 이름을 정의한 것이다.

반면 구현 모델인 WS-BPEL에서는 서비스 조합의 참여자를 아래와 같이 파트너 링크(partnerLinkType)와 포트타입(portType)으로 정의하여 사용한다.

이러한 두 모델 사이에 서비스 조합의 참여자에 대한 정보가 일치해야 한다. 이는 두 모델의 참여자가 실질적으로 구현된 서비스를 의미하는 것은 아니다. 실제 WS-CDL과 같은 코레어그래피 모델에서는 서비스의 위치 및 구현 내용을 포함하지 않는다. 그러나 WS-BPEL과 같은 구현 모델에서는 실제 동작을 위해 구현 관련 내용이 포함되어 있다. 따라서 두 모델 사이에서 나타나는 참여자의 경우 각각의 역할과 실제 상호작용을 위한 인터페이스가 동일한지 검증해야 한다. 참여 서비스의 일치 여부를 검증하기 위해서 우선 WS-CDL에서 정의하고 있는

(표 1) WS-CDL과 WS-BPEL의 데이터/메시지 포맷 대응 관계

WS-CDL	WS-BPEL	의미
<InformationType>	<Variable>	서비스 조합 내에서 사용하는 변수 및 데이터 타입
<VariableDefinition>		
<Token>	<Property>	데이터 타입의 일부 요소 및 프로세스 인스턴스의 상관관계
<TokenLocator>	<PropertyA-lias>	Token, Property의 별칭 또는 새로운 변수 할당

(표 2) WS-CDL과 WS-BPEL의 참여 서비스 요소 대응 관계

WS-CDL	WS-BPEL	의미
<participantType>	<PartnerLinkType>	조합에 참여하는 참여자 명시
<RoleType>	<Role>	조합 서비스들 사이의 참여자의 역할
<behavior>	<operation>	조합 서비스들 사이의 상호작용을 위한 인터페이스

```
<interaction name="createPO" channelVariable="tns: Shopping Mall Service-channel" opera-tion="PurchaseOrder">
      <participate relationshipType="tns: CustomerShoppingMallServiceRelationShip"
                   fromRole="tns: Customer" toRole="tns: ShoppingMallService"/>
      <exchange name="request" informationType="tns:purchaseOrder" action="request">
        <send variable="cdl:getVariable("tns:POMessage", "", "")" />
        <receive variable="cdl:getVariable("tns:purchaseOrderResultMessage", "", "")"
                 recordReference="record-the-channel-info" />
      </exchange>
      <exchange name="response" informationType=" purchaseOrderResultMessage" action="respond">
        <send variable="cdl:getVariable("tns:purchaseOrderAck", "", "")" />
        <receive variable="cdl:getVariable("tns:purchaseOrderAck", "", "")" />
      </exchange>
      <exchange name=" cannotCompleteOrder" informationType="InvalidCredit" action="respond">
        <send variable="cdl:getVariable(tns:badPurchaseOrderAck, "", "")" causeException="true" />
        <receive variable="cdl:getVariable("tns:badPurchaseOrderAck", "", "")"causeException="true" />
      </exchange>
</interaction>
```

(그림 9) WS-CDL에서 메시지 교환 정의

Customer	Shopping Mall Portal Service
<invoke name="purchaseOrder" suppressJoinFailure="yes" partnerLink="ShoppingMallService" portType="SP:purchaseOrderPT" operation="purchaseOrder" inputVariable="purchaseOrderMessage" outputVariable="purchaseOrderResultMessage"> <targets> <target linkName="ShoppingMallServiceLink" /> </targets> <catch faultName="SP: badPurchaseOrderAckException">...</catch> </invoke>	**<receive** partnerLink="Customer" portType=" purchaseOrderPT " operation=" purchaseOrder " variable=" purchaseOrderMessage " createInstance="yes" messageExchange="NCName"> <fromParts> <fromPart part="Customer" toVariable="purchaseOrderMessage" /> </fromParts> </receive> **<reply** partnerLink="Customer" portType="purchaseOrderPT" operation=" purchaseOrder" variable="purchaseOrderResultMessage" faultName="badPurchaseOrderAck Exception" messageExchange="NCName"> <toParts> <toPart part="Customer" fromVariable=" purchaseOrderMessage " /> </toParts> </reply>

(그림 10) WS-BPEL에서 메시지 교환 정의

participantType에 해당하는 Role Type 및 Behavior 정보를 추출하고 각각의 WS-BPEL에 정의된 PartnerLink의 Role과 PortType의 operation 내용과 비교한다. 이러한 참여 서비스의 일치 여부의 검증은 다음과 같이 요약할 수 있다.

3) 메시지 교환 프로토콜의 일치 여부

코레어그래피에서 메시지 교환 프로토콜은 참여하는 서비스들 사이에 메시지교환 방법과 순서를 명시한다. WS-CDL에서 이를 위해 사용되는 요소는 Interaction, relationshipType이다. Interaction은 참여하는 서비스들 사이에 정보를 주고 받는 방법을 명시하는 기본적인 구조로 호출되는 서비스 이름, 서비스 인터페이스, 역할 정보, 주고 받는 메시지 타입 및 데이터 타입을 이용한다. 또한 정보를 교환하는 과정에서 발생 가능한 예외 상황에 대해서도 이 Interaction을 통해 정의한다. 예를 들어, 다음과 같은 Interaction 요소는 Customer가 Shopping Mall Service에 특정 물품을 요청하는 과정에서의 메시지 교환 프로토콜을 포함하고 있다. 참여자는 Customer, Shopping Mall Service이며, PurchaseOrder 오퍼레이션을 통해 purchaseOrder 메시지를 보내고 결과로 purchaseOrder

ResultMessage를 받는 순서로 작업이 수행된다. 이 때 잘못된 요청에 대해서는 badPurchaseOrderAckException을 반환하여 에러 상황임을 알린다.

WS-BPEL의 경우 서비스를 요청하는 요청자의 invoke 활동과 서비스를 수행하는 측에서의 receive, reply 활동을 통해 메시지 교환 프로토콜이 정의된다. 예를 들어, 위의 상호작용에 대한 WS-BPEL 코드는 다음과 같다.

WS-CDL과 WS-BPEL에서의 메시지 교환 프로토콜의 일치 여부를 확인하기 위해서는 수행되는 오퍼레이션의 종류에 따라 각각 WS-CDL에서의 <Exchange>, WS-BPEL에서의 <Invoke>-<Receive><Reply>의 구조와 순서가 동일함을 검증해야 한다. 이를 위해서는 두 단계의 검증 과정이 수행되어야 한다. 우선 WS-CDL과 WS-BPEL에 기술된 단일 오퍼레이션에 대한 메시지 교환 방법에 대한 검증을 수행해야 하고, 두 번째로 전체 워크플로우에서의 메시지 교환 순서를 검증해야 한다. 첫 번째 단일 오퍼레이션에 대한 검증 방법은 (그림 11)의 프로시저와 같으며, 전체 워크플로우 부분은 다음 절의 워크플로우 및 제어 로직의 일치 여부 검증 방법을 통해 검증한다.

4) 워크플로우 및 제어 로직의 일치 여부

코레어그래피 기반 서비스 조합 모델과 이를 구현한 모델은 모두 서비스 조합에서의 메시지 상호작용을 정의하고 있다. 이러한 모델들은 아래와 같은 요소를 이용하여 서비스 조합에서의 제어 흐름과 메시지 교환 방법, 상호작용 등을 표현할 수 있다. 앞서 설명한 메시지 교환 프로토콜 검증이 개별 오퍼레이션에 대한 동일함을 판별하는 과정이었다면, 이 단계는 전체 워크플로우 관점에서의 검증 과정이다.

```
typedef     messageExchangeSequence
            //WS-CDL 오퍼레이션 메시지 시퀀스 저장을 위한 자료구조
typedef     validationResult
                        //검증 결과를 저장하기 위한 자료구조

// WS-CDL 파일
read        WS-CDL File
// WS-CDL 참여 서비스 역할 및 인터페이스 추출 및 저장
While (not End of WS-CDL)
    // Information Type 정의 부분 추출 및 저장
    get     Interaction from WS-CDL
    get     Interaction Sequence from WS-CDL
    save    Interaction Sequence to messageExchangeSequence
End While

//WS-BPEL 파일
For Each of WS-BPEL Files

    read BPEL File of FromService
    read BPEL File of ToService

    //WS-BPEL참여 서비스 역할 및 인터페이스 추출 및 저장
    While (not End of WS-BPEL File)
        get  Invoke Activity from FromService
        get  Receive Activity from ToService
        get  Reply Activity from ToService
        generate Invoke Sequence

        find Interaction Sequence info from messageExchangeSequence

        //참여 서비스 및 인터페이스 비교, 결과 저장
          if(find == TRUE)
                save valid to validationResult
            else
                save invalid to validationResult
        End While
End For
```

(그림 11) 메시지 교환 프로토콜의 일치 여부 검증 프로시저

(표 3)은 WS-CDL의 제어흐름 관련 요소이고, (표 4)는 WS-BPEL의 제어흐름 관련 요소들이다.

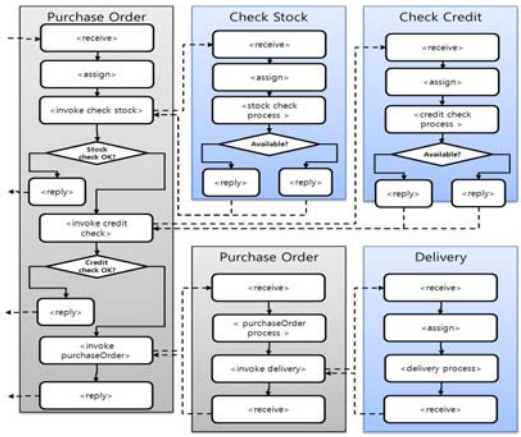
(표 3) WS-CDL의 주요 제어 흐름 요소

WS-CDL의 요소	의미
<workunit>	가드조건, 블록조건, 반복조건 정보를 나타내는 활동
<sequence>	일정한 순서로 호출되는 활동의 집합 정의
<parallel>	병렬적으로 호출되는 활동의 집합 정의
<choice>	여러 실행 경로 중 하나의 활동을 선택하여 실행
<interaction>	서비스의 호출 관계(상호작용)를 나타내는 활동
<perform>	실행 중인 코레어그래피 내에서 다른 코레어그래피를 호출하여 실행시키는 활동
<assign>	변수 할당 활동
<silentAction>	값 반환 등 겉으로 드러나는 행동을 하지 않는 활동
<noAction>	아무 행동도 하지 않음
<finalize>	완료 후의 최종 마무리 활동

(표 4) WS-BPEL의 주요 제어 흐름 요소

BPEL의 요소	의미
<receive>	클라이언트로부터의 요청을 수신
<reply>	동기화 작업의 응답 생성
<invoke>	다른 웹 서비스의 호출(상호작용 요청)
<assign>	데이터 변수의 값 할당
<throw>	폴트 및 예외 발생
<terminate>	전체 프로세스의 종료
<wait>	일정 시간 대기
<sequence>	일정한 순서로 호출되는 활동의 집합 정의
<switch>	분기 조건의 구현 (Case-switch 구조)
<while>	루프 정의
<pick>	다양한 실행 경로 중 하나를 선택
<flow>	병렬적으로 호출되는 활동의 집합 정의
<scope>	실행의 한 단위. 활동의 묶음을 표현하거나 특정 조건에 의해 내부의 활동을 수행
<empty>	아무 것도 하지 않음
<compensation-Handler>	모든 수행이 완료된 후 마무리하는 활동

본 논문에서는 전체 워크플로우 관점에서 코레어그래피 모델과 구현 모델 사이의 표현의 일치 여부를 검증하기 위하여 두 모델을 추상화된 그래프 형태로 표현하여 두 모델의 워크플로우와 제어 로직이 일치하는지를 검사한다. 코레어그래피 모델은 전역 모델로 모든 참여자에 대한 워크플로우와 제어 로직을 표현하는 반면, WS-BPEL 구현 모델은 각 참여자 별로 작성되기 때문에 두 모델이 일치한다는 것을 검증하기 위해서는 구현 모델을

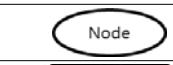
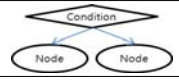
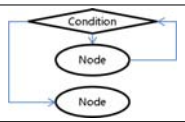


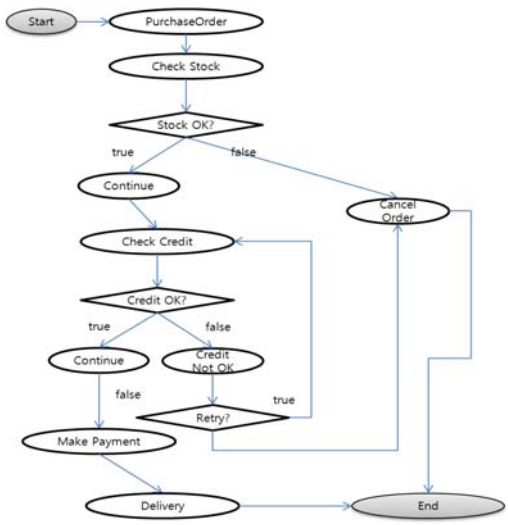
(그림 10) WS-BPEL기반 구현 모델

통합하는 과정이 필요하다. 예를 들어, (그림 6)의 코레어그래피를 WS-BPEL을 이용해 구현하는 경우 각 참여자들은 관련된 여러 서비스를 구현해야 한다. 제품의 구매와 관련된 서비스의 경우 (그림 10)과 같은 형태로 구현된다.

코레어그래피 모델의 제어흐름 그래프는 제어흐름에 대한 요소와 기본 블록(Basic Block)을 기반으로 제어흐름에 대한 요소만을 추출하여 전체 제어 흐름 그래프를 작성한다. 기본 블록은 특정 참여 서비스 내에서 순차적으로 수행되는 작업을 단일 노드(node)로 표현한다. 다른 참여 서비스에 접근하거나, 제어흐름과 관련된 부분은 에지(edge)로 표현한다. 제어 흐름과 관련된 요소로 분기의 경우 조건에 따른 여러 개의 노드와 에지를 이용하여 표현한다. (표 5)는 WS-CDL의 제어 요소에 대한 그래프

(표 5) WS-CDL의 제어흐름 요소 별 추상 그래프 생성 방법

WS-CDL의 요소	생성 방법 및 설명	표현
Sequence	Sequence 요소 내의 활동을 기본 블록으로 표현	
Parallel	병렬적으로 수행되는 여러 활동을 표현	
Choice	여러 실행 경로 중 하나의 활동을 선택 분기로 표현	
Interaction	서비스의 호출 관계(상호작용)를 표현 서비스 노드들 사이의 에지 연결로 표현	
Loop	WS-CDL에서는 Loop에 대한 표현은 없으나 워크플로우의 흐름상 반복적인 흐름이 존재함	



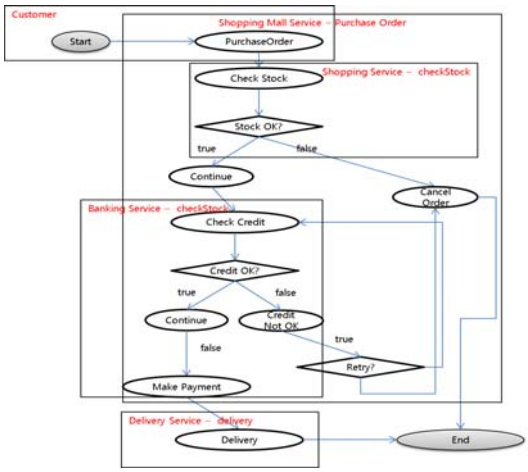
(그림 11) WS-CDL 예제에 대한 추상 그래프

표현 방법의 대응 규칙을 나타낸 것이다. 제어 흐름 및 워크플로우와 관련이 없는 요소는 고려하지 않는다.

(그림 11)은 (그림 6)의 코레어그래피 예제를 추상화 그래프로 표현한 것이다.

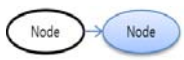
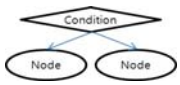
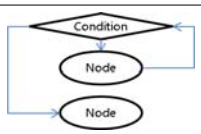
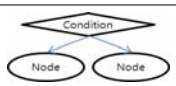
WS-BPEL모델의 경우 서비스 조합에 참여하는 각각의 서비스에 대해 WS-BPEL 혹은 단일 서비스 형태로 구현되기 때문에 각 서비스 내의 제어흐름을 분석하여 추상화된 그래프 형태를 추출한다.

WS-BPEL의 경우 여러 개의 파일로 구성된다. 따라서 각각의 파일에 대한 추상 그래프를 생성하는 과정과 이를 통합하는 과정을 통해 최종 추상 그래프를 생성해야 한다. 예를 들어, purchaseOrder 서비스에 대한 각 참여 서비스들을 개별적인 추상 그래프로 생성하고, <Invoke> 활동을 통해 호출되는 서비스를 에지로 연결함으로써 WS-CDL과 같은 전역 모델 형태의 추상 그래프가 생성된다. (그림 12)는 이러한 통합 결과에 대한 추상 그래프를 보여준다.



(그림 12) WS-BPEL 그래프의 통합

(표 6) WS-BPEL의 제어흐름 요소 별 추상 그래프 생성 방법

BPEL의 요소	생성 방법 및 설명	표현
<receive> <reply> <invoke>	서비스들 간 상호작용을 표현 노드의 연결로 표현	
<sequence>	내부의 순차적인 제어흐름, 기본 블록으로 표현	
<switch>	조건을 갖는 분기와 연결된 여러 노드로 표현	
<while>	조건을 갖는 분기, 노드 사이의 연결로 표현	
<pick>	조건을 갖는 분기와 연결된 여러 노드로 표현	
<flow>	동시에 병렬적으로 수행되는 여러 활동을 표현	

(표 7) WS-CDL, WS-BPEL 비교를 위한 요소 맵핑

WS-CDL 요소	WS-BPEL 요소
<workunit>	<scope>
<repeat>	condition in a <while> loop
<guard>	<condition>
<sequence>	<sequence>
<parallel>	<flow>
<choice>	<case> in <switch>
<interaction> action request	<invoke>
	<receive>
<interaction> action response	<invoke>
	<receive>
<timeout>	<pick>
<assign>	<assign>
<slientaction>	<empty>
<noaction>	Empty
<finalize>	<compenstionHandler>

(그림 11)과 (그림 12)는 각각 WS-CDL과 WS-BPEL로 표현된 추상 그래프이다. 이 두 그래프의 비교를 통해 제어 흐름 및 워크플로우가 동일한지 여부를 검증할 수 있다. 일반적으로 두 그래프의 일치 여부를 판단하는 것은 매우 어려운 문제이지만 여기서는 그래프의 각 노드가 명시적으로 구분되고 또한 이름을 갖고 있으므로 각각의 노드와 에지의 대응 관계를 파악하는 방법을 통해 두 그래프의 일치 여부를 확인할 수 있다. 따라서 본 논문에서는 두 그래프의 시작 노드에서부터 종료 노드까지 각각의 노드와 에지를 순차적으로 비교하여 두 그래프가 일치하는지 판단한다. 이 때 실질적인 그래프 요소의 비교는 다음의 맵핑 관계를 통해 수행한다.

3.3 코레어그래피 기반 서비스 조합 기능 테스트

코레어그래피 기반 서비스 조합 모델이 구현 모델과 일치하는 것을 검증하였다 하더라도 구현된 기능이 의도한 대로 정확히 동작한다고 보장할 수 없다. 이는 코레어그래피 모델 자체에 오류가 있거나 내부 로직의 구현이 잘못된 경우 제어 흐름이 일치한다 하더라도 실제 동작에서 잘못된 기능을 수행할 수 있기 때문이다. 따라서 코레어그래피 모델의 실제 구현된 시스템이 정확히 동작하는지 검증할 필요가 있다. 본 논문에서는 이를 테스트하기 위해 코레어그래피 기반 테스트 케이스 생성 방법과

(표 8) 테스트 되어야 할 경로

checkStock	checkCredit	Credit Check Re-try
F	-	-
T	T	-
T	F	T
T	F	F

실제 구현 모델에서의 테스트 수행 방법 및 테스트 수행 결과 판단 방법을 제시한다. 우선 코레어그래피 모델을 기반으로 제어 로직을 분석하여 테스트 되어야 할 경로 정보를 생성한다. 테스트 수행은 실제 구현된 시스템에 탐침을 삽입하여 테스트 케이스가 동작할 때 정확한 동작을 수행하는지 확인한다. 마지막으로 테스트 결과의 분석은 코레어그래피 기반 서비스 조합에서의 기능 수행 모델과 실제 구현 모델 사이의 경로 분석을 통해 수행한다. 각각의 방법에 대한 구체적인 사항은 다음과 같다.

1) 테스트 대상 경로 정보 파악

코레어그래피 형태의 웹 서비스 조합에서는 각 서비스들의 상호작용이 원활하게 이루어지는 지를 테스트 하는 것이 중요하다. 서비스 일부분에 대한 테스트가 아니라 웹 서비스 조합 전체의 흐름이 의도한 대로 실행되는지를 테스트하기 위해서는 전역 모델을 대상으로 테스트할 필요가 있으며, 이를 위해 본 논문에서는 경로 커버리지(path coverage)를 기준으로 테스트를 수행하고자 한다.

경로 테스트의 목적은 로직을 이루고 있는 구성 요소들 간의 상호 독립적인 경로를 모두 수행해 봄으로써, 잠재적인 오류를 찾아내는 것이다. 프로그램 내의 모든 독립적인 경로를 수행하게 함으로써 모든 경로를 탐색할 수 있으며, 일반적으로 사용되지 않는 조합에서 발생할 수 있는 오류도 검출 할 수 있다. 이러한 경로 테스트를 위해서는 프로그램 내부의 수행 흐름에 대한 분석을 수행해야 한다. 코레어그래피 기반 서비스 조합의 경우 워크플로우 및 제어 로직의 분석을 통해 웹 서비스 조합에서의 경로를 분석하여 테스트하는 것이 가능하다. (그림 11)의 추상 그래프의 경우 두 개의 선택과 하나의 반복 구조를 갖는 제어 흐름을 포함하고 있다. 이러한 제어 흐름에서 선택의 경우 True/False를 모두 커버할 수 있도록 테스트 케이스를 선정하여야 하고, 반복의 경우 반복 부분을 한번도 수행하지 않는 경우와 한번 이상 수행하도록 테스트 케이스를 선정하여야 한다. (그림 11)의 그래프를 대상으로 100%의 경로 커버리지를 만족할 수 있도록 테스트를 수행하기 위해서는 다음과 같은 경로들을 테스트하여야 한다.

(표 9) 테스트 데이터 생성

Service	T/F	Test Data 요구사항	Message type
checkStock	True	상품번호: 10자리 정수형 숫자 재고 수: 5자리 정수형 숫자 조건: 상품번호와 재고수의 형태가 일치하고, 상품 번호에 따른 재고수는 주문 요청한 숫자보다 많아야 한다.	POMessage/ StockOK
	False	조건: 상품 번호에 따른 재고수가 주문 요청한 숫자보다 적음.	OutOfStockException
checkCredit	True	카드 번호: 16자리 정수형 숫자 비밀 번호: 4자리 정수형 숫자 조건: 두 번호의 형태가 일치하고, 카드 번호에 대응하는 비밀번호가 일치해야 함.	CustomerInfo/ CreditOK
	False	조건: 카드 번호에 대응하는 비밀 번호가 일치하지 않음	InvalidCredit
Credit Check Retry	True	조건: CheckCredit이 False이며 새로운 카드정보로 재시도	CreditOK
	False	조건: CheckCredit이 False이며 새로운 카드 정보로 재시도하지 않음	InvalidCredit

2) 테스트 수행

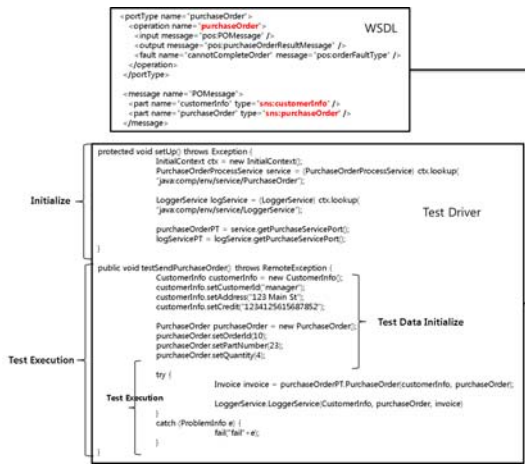
① 테스트 데이터 생성

WS-CDL의 분석을 통해 서비스 조합에서 테스트 되어야 할 경로 정보가 주어지면 경로를 테스트하기에 적합한 테스트 데이터를 생성할 수 있다.

checkCredit에서 참이 되는 경우는 전달되는 메시지값이 "CreditOk"인 경우이며 이 메시지 값들은 WS-CDL 구성 단계에서 <InformationType> 요소에서부터 정의되어 있다. <interaction>의 하위 요소인 <ex-change>에서 상호작용이 발생하고 메시지 교환 시에 어떤 <InformationType>을 가지고 있는 지를 보이므로 WS-CDL의 <interaction>, <exchange>, <informationType> 요소를 분석함으로써 메시지 값을 알아낼 수 있고 이 메시지들이 테스트 데이터가 된다.

② 테스트 드라이버 생성

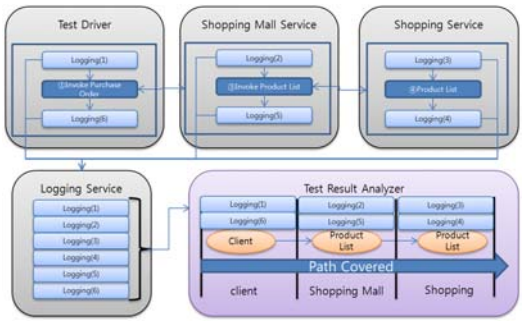
테스트 드라이버는 시스템 및 시스템 컴포넌트를 테스트하는 환경의 일부분으로 테스트 대상 소프트웨어를 구동하기 위한 코드를 말한다. WS-CDL을 통해 추출된 정보는 실행 가능한 서비스 조합에 대한 정보가 아니기 때문에 실제 수행과 관련된 BPEL로부터 테스트 드라이버 생성에 관련된 정보를 획득한다. 이를 위해 조합 서비스의 WSDL 파일로부터 필요한 정보를 추출한다. 본 논문의 예제에서 실제 테스트 수행 대상인 조합서비스는 고객이 접근하는 Shopping Mall Portal Service이다. 따라서 이 서비스의WSDL을 분석하여 테스트 드라이버를 생성한다. 예를 들어 purchaseOrder에 대한 WSDL파일과 생성된 테스트 드라이버의 형태는 (그림 13)과 같다. 이렇게 생성된 테스트 드라이버는 이미 생성된 테스트 데이터와 같이 테스트 스위트(Test Suite)로 묶여 사용된다.



(그림 13) 테스트 드라이버 생성

③ WS-BPEL 확장, 테스트 결과 분석, 테스트 커버리지 분석

WS-BPEL확장은 웹 서비스 조합이 테스트 되는 과정에서 중간 테스트 결과를 저장하기 위해 WS-BPEL 파일을 확장하는 것을 의미한다. 이는 웹 서비스 조합의 상호

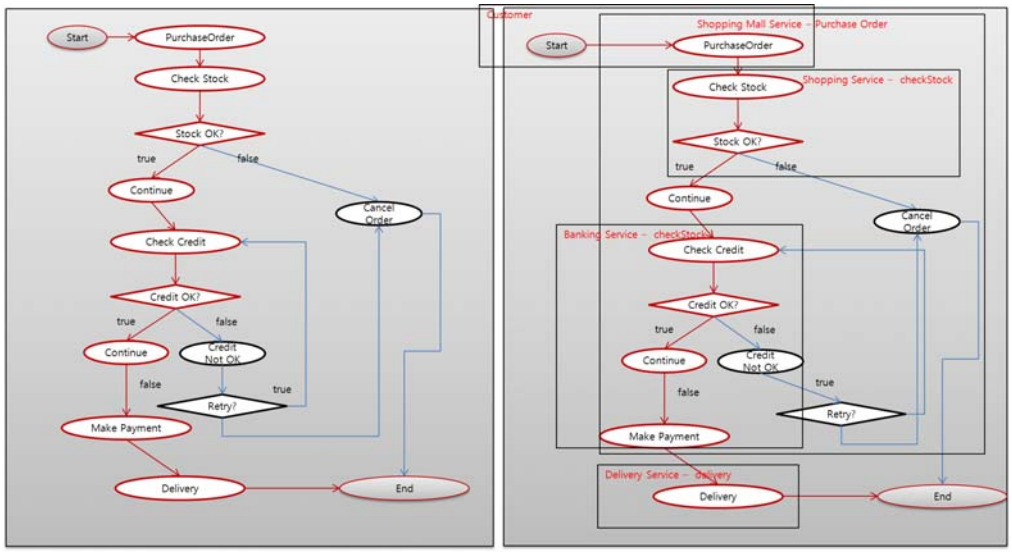


(그림 14) 로그 서비스, BPEL 확장, 테스트 결과 분석 과정

작용이 일어나는 부분에 탐침(probe)을 삽입하여 실제 그 실행 경로를 지나는 지를 확인하고 기록하기 위해 필요한 기능이다. 본 논문에서는 WS-BPEL의 기능에 미치는 영향을 최소화 하고, 테스트 수행과정에서 발생하는 모든 상황을 기록하기 위해 외부의 로그 서비스를 이용한다. 서비스 조합의 실행 과정에서 발생하는 이벤트, 오류, 예외상황 등에 대한 정보를 시간과 호출 순서를 고려하여 기록한다. 향후 이 정보를 통해 의도했던 경로로 수행되었는지를 확인할 수 있다. 탐침이 삽입되는 위치는 서비스가 호출되기 이전, 호출 이후 그리고 예외 상황이 발생하는 위치이다.

(그림 14)는 이러한 테스트 결과 분석을 위한 과정을 나타낸다. 각 서비스는 내부의 로직 수행과 외부의 서비스의 호출 이전, 이후에 로그 서비스를 호출하여 테스트 수행 중간 결과를 저장한다. 로그 파일의 내용을 통해 조합된 서비스가 어떤 경로로 수행되었는지 알 수 있다. 로그 파일의 분석결과를 이용해 코레어그래피(WS-CDL) 모델과 구현 모델(WS-BPEL)의 추상 그래프에 수행된 경로를 표시하면 (그림 15)와 같다.

본 논문에서 제시하는 웹 서비스 조합의 테스트 방법은 코레어그래피 모델을 기반으로 테스트 케이스와 테스트 환경을 자동으로 생성한다. 따라서 전체 참여하는 서비스들 사이의 서비스 조합 구현이 코레어그래피 모델과 일치하는지 여부를 판별할 수 있다. 또한 내부의 제어 로직을 기반으로 경로 기반 테스트를 수행할 수 있기 때문에 서비스 조합 내의 로직 오류, 산술 오류, 인터페이스 오류를 검출할 수 있다. 예를 들어 인터페이스 오류의 경우 조합된 서비스의 인터페이스를 잘못 구현하더라도 초기 모델링 된 코레어그래피를 기반으로 테스트 케이스를 생성하기 때문에 이를 충분히 검출할 수 있다. 또한 테스트 수행과 결과가 웹 서비스 조합 그래프 모델의 경로를 기반으로 하기 때문에 로직 오류를 검출할 수 있다. 마지막으로 산술 오류의 경우 테스트 수행과정에서 나타나는 변수 값의 로그를 통해 검출할 수 있으며, 산술 오류의 결과로 잘못된 경로의 로직이 수행되는 경우 역시 검출할 수 있다. 그러나 본 논문에서 제시하는 방법이 얼마나



(그림 15) 실행 경로 분석 결과

효과적으로 오류를 검출하는지에 대한 평가는 여러 코레어그래피 기반 웹 서비스 조합을 테스트한 후 정량적인 평가를 통해 이루어져야 할 것이다. 따라서 향후 연구로 본 논문에서 제안하는 테스트 방법을 다양한 서비스 조합에 적용할 예정이다.

4. 결론 및 향후 연구 방향

코레어그래피는 다수의 기업이 참여하는 복잡한 서비스의 조합에서 전체 비즈니스 프로세스의 흐름을 정의하기 위해 사용되며, 단순한 서비스의 시작점이나 예상되는 마지막 지점이 아닌 거시적인 측면에서 전체 프로세스를 모델링하기 위해 사용한다. 그러나 이러한 코레어그래피는 실질적인 비즈니스 프로세스의 수행을 지원하지 않기 때문에 오케스트레이션에 비해 그 활용도가 많지 않다. 또한 코레어그래피를 기반으로 정의된 비즈니스 프로세스에 대한 적합성 검증 및 테스트에 대한 노력도 많이 수행되지 않았다. 일반적으로 코레어그래피 기반 웹 서비스 조합에서는 코레어그래피로 작성된 서비스 조합 모델과 실제 구현된 모델의 제어흐름이 일치하는지 여부와 코레어그래피로 정의된 서비스 조합이 실제 동작 과정에서 문제없이 수행되는지 테스트하는 방법을 통해 코레어그래피로 작성된 서비스 조합의 기능을 검증한다. 이에 본 논문에서는 WS-CDL를 통해 정의된 코레어그래피 기반 웹 서비스 조합의 테스트 방법을 제시한다. 이를 위해 코레어그래피 모델과 구현 모델 사이의 구현 적합성 검증 방법과 구현된 웹 서비스 조합의 기능 테스트 방법을 제시한다.

코레어그래피 모델과 구현 모델 사이의 적합성을 검증하기 위해서 데이터 타입 및 메시지 포맷의 일치 여부, 참여 서비스의 일치 여부, 메시지 교환 프로토콜의 일치 여부, 워크플로우 및 제어 로직의 일치 여부를 검증하는 방법을 통해 거시적 관점에서의 코레어그래피 모델과 구현 모델의 일치 여부를 확인하는 방법을 제시하였다. 또한 이러한 워크플로우 및 제어 로직을 기반으로 경로 커버리지 기준을 만족하는 테스트 케이스를 생성하고, 실제구현 모델에서의 동작을 확인할 수 있도록 테스트 방법을 제안하였다. 테스트 방법은 WS-CDL과 WS-BPEL로부터 테스트에 필요한 정보를 추출하고 이를 기반으로 테스트 드라이버 및 테스트 데이터를 생성한다. 이러한 테스트 드라이버와 테스트 데이터를 이용해 테스트 수행과 테스트결과를 판별 할 수 있는 방법을 제시하였다.

본 논문의 두 가지 관점에서의 테스트 방법은 기존의 단일 웹 서비스 및 오케스트레이션 기반 웹 서비스 조합에서의 테스트 보다 거시적 관점에서의 테스트 수행을 지원한다. 이러한 거시적 관점의 웹 서비스 조합에 대한 테스트 수행 방법 및 테스트 환경을 제시함으로써 코레어그래피 기반 웹 서비스에 대한 검증에 널리 활용될 수 있을 것이다. 향후 본 논문에서 제시한 테스트 수행 방법 및 테스트 프레임워크를 이용해 실질적인 코레어그래피 기반 웹 서비스 조합에 대한 더 많은 테스트 작업을 수행할 예정이다.

참 고 문 헌

[1] M. P. Papazoglou et al. "Service-Oriented Computing," Communications of ACM, Vol. 46, No. 10, pp.25-28, 2003

[2] w3c, Web Services Architecture, http://www.w3.org/TR/2003/WD-ws-arch-20030808/, 2003.08

[3] C. Peltz, "Web services orchestration and choreography," IEEE Computer, Vol.36, No.8, pp.46-52, 2003

[4] W3C, Web Services Choreography Description Language Version 1.0, http://www.w3.org/TR/ws-cdl-10, 2005

[5] OASIS, Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007

[6] π4 Technologies Foundation, pi4soa2.1.0.GA, http://pi4soa.sourceforge.net, 2009

[7] A. Bertolino et al. "The audition framework for testing web services interoperability". In EUROMICRO-SEAA, pp.134-142, 2005

[8] P. Mayer et al., "Towards a BPEL unit testing framework". In TAV-WEB '06: Proceedings of Workshop on Testing, analysis, and verification of web services and applications, pp.33-42. ACM Press, 2006.

[9] Y. Yuan, et al. "A graph-search based approach to bpel4ws test generation." In Proceedings of the International Conference on Software Engineering Advances (ICSEA2006), IEEE Computer Society, 2006.

[10] H. Huang et al. "Automated model checking and testing for composite web services". In ISORC, pp.300-307, 2005.

◐ 저 자 소 개 ◑

국 승 학 (Seung Hak Kuk)

2004년 충남대학교 컴퓨터과학과 졸업(학사)
2006년 충남대학교 컴퓨터공학과(공학석사)
2012년~ 충남대학교 컴퓨터공학과(공학박사)
관심분야 : 소프트웨어 테스팅, 소프트웨어 품질관리, SOA, 웹 서비스
E-mail : triple888@cnu.ac.kr

서 용 진 (Yong Jin Soo)

2011년 충남대학교 컴퓨터공학과 졸업(학사)
2011년~현재 충남대학교 컴퓨터공학과 석사 재학
관심분야 : 소프트웨어 테스팅, 소프트웨어 품질관리, 스마트폰, UX/UI
E-mail : yjseo082@cnu.ac.kr

김 현 수 (Hyeon Soo Kim)

1988년 서울대학교 계산통계학과 졸업(학사)
1991년 한국과학기술원 전산학과(공학석사)
1995년 한국과학기술원 전산학과(공학박사)
1995년~1995년 한국전자통신연구원 Post Doc.
1996년~2001년 금오공과대학 조교수
1999년~2000년 Colorado State University 방문 연구 교수
2007년~2008년 Purdue University 방문 연구 교수
2001년~현재 충남대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 공학, 소프트웨어 테스팅, SOA
E-mail : hskim401@cnu.ac.kr