SOA에서 서비스 조합의 강건성 테스트 방법 및 테스트 프레임워크

(A Robustness Test Method and Test Framework for the Services Composition in the Service Oriented Architecture)

국 승 학 † 김 현 수 ††
(Seung Hak Kuk) (Hyeon Soo Kim)

요 약 최근 웹 서비스 기반 서비스 지향 구조는 네트워크 상에 분산된 다양한 애플리케이션을 효과적으로 통합하기 위한 방법으로 널리 활용되고 있다. 서비스 지향 구조에서 BPEL은 비즈니스 프로세스 모델 언어로, 다양한 서비스들을 통합하는 방법을 제시하고 있다. 현재 이러한 BPEL을 이용한 서비스 통합 및 조합에 관해 많은 연구가 진행되고 있으며 서비스들 간의 호환성을 검증하려는 노력, 비즈니스 프로세스의 식별과 추적에 관한 몇몇 연구가 진행되었다. 그러나 다양한 서비스의 조합으로 인해 발생하는 문제를 해결하려는 연구는 부족하다. 특히 조합된 서비스가 얼마나 신뢰할 수 있는지, 예외 상황에 대해 얼마나 강건하게 대처할 수 있는지 평가하고자 하는 노력은 거의 이루어지지 않았다. 이에 본 논문에서는 BPEL을 이용한 서비스 조합에 있어서 조합된 서비스의 강건성을 테스트하기 위한 방법과 이 방법을 지원하기 위한 테스트 프레임워크를 제시한다. 본 논문의 방법은 BPEL 프로세스와 참여하는 다양한 서비스를 분석하고, 분석된 정보를 바탕으로 실제 서비스들에서 발생 가능한 다양한 예외 상황을 발생시키는 가상의 환경을 구축하여 강건성 테스트를 수행한다. 이는 BPEL 프로세스로 표현된 서비스 조합이 얼마나 예외 상황에 강건하게 대처하는지 검증하는 방법이다.

키워드 : 서비스 지향 아키텍처, BPEL, 서비스 조합, 강건성 테스트, 테스트 자동화

Abstract Recently, Web services based service-oriented architecture is widely used to integrate effectively various applications distributed on the networks. In the service-oriented architecture BPEL as a standard modeling language for the business processes provides the way to integrate various services provided by applications. Over the past few years, some types of studies have been made on testing compatibility of services and on discriminating and tracing of the business processes in the services composition. Now a lot of studies about the services composition with BPEL are going on. However there were few efforts to solve the problems caused by the services composition. Especially, there is no effort to evaluate whether a composite service is reliable and whether it is robust against to exceptional situations. In this paper, we suggest a test framework and a testing method for robustness of the composite service written in WS-BPEL. For this, firstly we extract some information from the BPEL process and the participant services. Next, with the extracted information we construct the virtual testing environment that generates various faults and exceptional cases which may be raised within the real services. Finally the testing work for robustness of a composite service is performed on the test framework.

Key words : Service Oriented Architecture, BPEL, Service Composition, Robustness Test, Test Automation

· 이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-521-D00400)

† 정 회 원 : 충남대학교 컴퓨터공학
triple888@cnu.ac.kr
†† 종신회원 : 충남대학교 컴퓨터공학 교수
hskim401@cnu.ac.kr
(Corresponding author임)

논문접수 : 2009년 7월 24일
심사완료 : 2009년 8월 14일

정보과학회논문지: 소프트웨어 및 응용 제36권 제10호(2009.10)

1. 서 론

서비스 지향 구조(SOA: Service Oriented Architecture)는 잘 정의된 인터페이스와 서비스들 간 계약을 통해, 서비스라고 하는 애플리케이션의 다양한 기능 단위를 상호 연관시키는 컴포넌트 모델이다[1]. 서비스는 하드웨어 플랫폼, 운영체제, 프로그래밍 언어에 독립적인 방식으로 정의되며, 다양한 이기종 시스템들에 구현된 어떤 서비스라도 일반적이고 통합된 방식으로 상호작용할 수 있다. 오늘날의 서비스 지향구조는 대부분 XML 기반의 웹 서비스(Web Services)로 구현되고 있다. 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 서비스 지향 구조를 가장 잘 구현하고 있는 것이 웹 서비스이다[2].

웹 서비스는 분산된 애플리케이션을 서비스로 구축하고, 검색하며, 접근하는 방법에 대해 다루고 있다. 그러나 이러한 웹 서비스 기술들 만으로 서비스 지향 구조가 완성되는 것은 아니다. 서비스 지향 구조에서는 다양한 애플리케이션의 조합을 통해 새로운 서비스를 창출할 수 있는 방법도 또한 포함되어야 한다. 이를 위해서는 다양한 서비스들을 엮어 하나의 비즈니스 프로세스로 표현하고, 서비스들 간의 워크플로우를 정의하기 위한 수단이 필요하다. 서비스 지향 구조에서 다양한 서비스 조합(services composition)을 표현하기 위한 대표적인 수단으로 BPEL(Business Process Execution Language)이 있다[3]. BPEL은 비즈니스 프로세스를 기술하는 고수준의 언어로 참여하는 서비스들 간의 상호작용을 워크플로우 형태로 기술할 수 있다. BPEL은 조합에 참여하는 웹 서비스들간의 상호작용을 다양한 요소를 이용해 비즈니스 프로세스의 형태로 표현할 수 있어서 서비스 기반 정보시스템 통합 분야에 널리 활용되고 있다. 그러나 지금까지 BPEL과 관련된 연구는 프로세스의 식별과 추적, 그리고 서비스들 간의 호환성 문제에 초점이 맞춰져 있고, 다양한 예외 상황에 대해 강건한 서비스 조합을 구축하고자 하는 노력은 이루어지지 않았다.

BPEL 프로세스에 의한 서비스 조합은 참여하는 개별 서비스의 에러, 개별 서비스의 예외 상황, 기술된 프로세스의 오류, 제공되는 환경의 오류 등으로부터 영향을 받는다. 따라서 BPEL 프로세스는 이러한 다양한 오류 및 예외 상황을 처리할 수 있도록 기술되어야 하며 이를 확인할 수 있는 방안이 필요하다. 그러나 이러한 서비스의 조합을 테스트하는 것은 기존의 여타 애플리케이션을 테스트하는 것 보다 어렵다. 그 이유는 참여하는 서비스의 수에 따라 다양한 테스트 시나리오가 작성되어야 하고, 또한 만일 상용 서비스를 이용해야 할 경우 테스트 수행 비용이 증가하기 때문이다. 따라서 제약된 시간과 비용으로 보다 더 많은 테스트를 수행하기 위해서는 새로운 테스트 방법이 요구된다. 이에 본 논문에서는 BPEL 기반의 서비스 조합의 강건성을 테스트하기 위한 방법과 이 방법을 지원하기 위한 테스트 환경을 제안한다. 제안하는 강건성 테스트 방법은 BPEL 프로세스와 서비스 조합에 참여하는 서비스를 분석하여 프로세스 실행 중 고려해야 할 다양한 오류 상황을 발생시키는 가상의 서비스 환경을 생성하고, 이를 통해 기술된 BPEL 프로세스가 얼마나 오류상황과 예외 상황에 강건하게 대처할 수 있는지를 테스트 한다. 또한 이 테스트 방법을 검증하기 위하여 테스트 프레임워크에 대한 프로토타입을 구축하여 적용한 사례도 제시한다.

본 논문의 구성은 다음과 같다. 2절에서는 서비스 지향 구조와 서비스 조합에 대해 살펴보고, 기존의 서비스 조합과 관련된 테스트 연구들을 살펴본다. 3절에서는 서비스 조합에서의 강건성에 대해 기술하고, 본 논문이 제시하는 강건성 테스트 방법에 대해 기술한다. 이를 위해 일반적인 테스트 프로세스에 맞추어 각 단계별 테스트 활동을 설명한다. 4절에서는 본 논문의 테스트 방법을 지원하기 위한 테스트 프레임워크에 대해 설명하며, 5절에서는 e-엔지니어링 프레임워크에서 펌프 설계 애플리케이션을 대상으로 본 논문의 테스트 방법을 적용한 사례연구를 소개한다. 마지막으로 6절에서 결론과 향후 연구 방향에 대해 기술한다.

2. 관련연구

2.1 서비스 지향 구조 및 웹 서비스

서비스 지향 구조(SOA)에서는 애플리케이션의 기능들을 사용하기에 적합한 크기로 공개한 서비스들과 이의 제공, 사용에 관한 정책이나 적용 방법을 정의할 수 있다. 즉 서비스라 불리는 분할된 애플리케이션 조각들을 단위로 이들을 약결합으로 연결하여 하나의 완성된 애플리케이션을 개발하기 위한 소프트웨어 아키텍처라 할 수 있다[4]. 이러한 SOA는 소프트웨어 아키텍처로써 기술로부터 독립적인 반면, 웹 서비스는 이러한 아키텍처를 실현하기 위한 기술들의 모음이라고 할 수 있다. 웹 서비스는 HTTP, XML, SOAP, WSDL, UDDI 등을 기본 구조로 이용하고 있으며 이들 기술은 특정 플랫폼에 독립적이며, 표준화 되어 있고, 또한 상호 운용성을 보장한다. 웹 서비스는 크게 서비스 작성과 기술(description), 그리고 등록, 발견, 호출로 구성된 개발단계를 가지고 있으며, 각 개발단계의 지원을 위하여 관련 표준들을 이용하고 있다.

2.2 웹 서비스 조합

웹 서비스는 다른 웹 서비스를 호출하지 않고 자신의 서비스 실행 결과만 반환하는 단일 서비스(atomic service)와 하나의 서비스 내에서 프로세스 실행 흐름에 따라 외부의 다른 서비스를 호출하는 조합 서비스(composite service)로 구분할 수 있다. 웹 서비스 조합이란 복합 서비스를 개발하는 일련의 과정으로써 웹 서비스 참여자의 웹 서비스들을 조합하여 비즈니스 프로세스로 구성하고, 프로세스 논리대로 웹 서비스를 실행시키는 메커니즘이다. 웹 서비스 조합 방식으로는 코리오그래피(choreography)와 오케스트레이션(orchestration)이 있다[5]. 오케스트레이션은 어떤 한 파트너의 내부 비즈니스 프로세스 논리를 의미하며, OASIS(Organization for the Advancement of Structured Information Standards)에서 정의한 BPEL이 사용된다. 코리오그래피는 두 파트너 사이에서 일어나는 메시지 교환 방식을 의미하며, 일반적으로 둘 이상의 웹 서비스 참여자가 상호작용을 할 때 생길 수 있는 웹 서비스 프로세스로써 W3C의 WS-CDL(Web Services Choreography Description Language)로 표현된다.

2.3 웹 서비스 조합 테스트에 관한 기존 연구

웹 서비스 조합을 테스트하기 위한 기존의 연구들이 몇 가지 존재한다. [6]의 경우 오케스트레이션 기반의 웹 서비스 조합을 위한 화이트 박스(white-box) 단위 테스트 프레임워크를 제안하였다. 일반적으로 오케스트레이션은 오케스트레이터에 의해 실행 가능한 비즈니스 프로세스를 기술하고 이를 실행한다. 이 논문에서 제안하는 프레임워크는 이러한 비즈니스 프로세스를 하나의 조합된 서비스로 보고 해당 프로세스의 실행 흐름에 따라 테스트를 수행한다. [7]도 오케스트레이션에서의 단위 테스트를 위한 화이트 박스 테스트 방법을 제안하였다. [6]에서는 테스트 케이스 생성에 관한 점을 고려하지 않은 반면 [7]에서는 제어 흐름 커버리지 기준(control flow coverage criteria)을 적용한 테스트 케이스 생성 방법에 대해 제안하였으며, 제어 및 데이터 흐름을 통한 모델 체킹(model checking) 도구를 제안하였다. 그러나 [6], [7] 논문의 경우 BPEL 프로세스로 기술된 서비스 조합의 기능 테스트를 수행하기 위하여 BPEL 프로세스의 제어흐름을 기반으로 테스트 케이스를 생성하기 때문에 예외적인 상황에 대한 대처 능력을 테스트하기 어렵다. 사실 이 연구들은 테스트의 목적에서 우리 연구와 차이가 있으며, 본래의 목적인 기능 테스트를 확장해서 강건성 테스트를 수행한다 하더라도, 제어흐름에 나타나지 않는 예외적인 상황을 테스트 할 수 없기 때문에 강건성 테스트에 적용하기 어렵다. [8]에서는 OWL-S를 새로운 프로세스로 자동으로 변환해 주는 통합 프로세스를 제안하였다. 이 과정에서 기존의 모델 체킹 도구인 BLAST를 이용하여 정의된 비즈니스 프로세스를 검증하는 방법을 제시하였다. [9]에서는 코리오그래피 기반의 서비스 조합을 테스트하기 위한 확장된 UDDI 레지스트리를 제안하였다. 확장된 UDDI는 코리오그래피에 참여하는 개별 서비스들을 UDDI에 등록할 때 새로운 서비스가 조합에 적합한지 검증하는 기능을 수행한다. 논문 [9]의 경우 조합 적합성을 검증하기 위해 UDDI를 이용하였지만, 그 검증 대상이 WSDL에 기술된 인터페이스를 대상으로 검증을 수행하기 때문에 실제 서비스가 수행되는 동안 발생할 수 있는 다양한 문제에 대처할 수 있는 능력인 강건성에 대한 테스트 방법을 제시하지 않는다.

또한 논문 [10]의 경우 단일 웹 서비스의 강건성 테스트 방법을 제시하고 있다. 이는 웹 서비스의 WSDL 정보를 기반으로 테스트 환경을 구축하고, 테스트 코드와 테스트 데이터를 자동으로 생성하는 등 테스트 과정을 자동화하는 방법을 설명한다. 논문 [11]의 경우 [10]의 연구와 유사한 방법으로 웹 서비스의 기능 테스트를 수행한다. 이 연구에서 역시 WSDL을 분석하여 테스트에 필요한 정보를 추출한다. 그러나 보다 더 복잡한 데이터 타입을 처리하기 위해 WSDL에 기술된 데이터 타입 정의 부분을 분석하고 그에 맞는 테스트 데이터를 자동으로 생성한다. 이들 두 연구에서는 강건성 테스트를 위해서 테스트 데이터 생성 과정에서 예외적인 상황을 유발하는 테스트 데이터를 생성함으로써 웹 서비스가 이러한 데이터에 대해 강건하게 대처하는지 확인한다. 그러나 이 방법들은 개별 서비스의 예외처리 방법을 적절하게 테스트 할 수 있으나 조합된 서비스에 대한 예외처리 능력을 테스트하지 못한다. BPEL로 기술된 서비스 조합의 경우 개별 서비스의 예외 상황뿐만 아니라 서비스들 간의 상호의존성에 의해서 발생하는 다양한 예외 상황과 또한 서비스 수행 환경으로 인해 발생하는 예외 상황 등이 존재하는데 이들을 모두 테스트할 수 있어야 하기 때문이다.

위와 같이 기존의 테스트 노력들은 서비스 조합의 기능적인 측면에서 단위 및 통합 테스트를 위한 테스트 케이스 생성, 모델 체킹 등을 통한 프로세스의 검증, 그리고 서비스 조합의 적합성 테스트에 대해 초점을 맞추고 있다. 그러나 서비스 조합에 있어서 조합된 서비스가 다양한 예외 상황에 대해 강건하게 대처할 수 있는지 확인하는 방법에 대한 연구는 거의 수행되지 않았다. 대부분의 경우 개별 서비스 및 조합된 서비스의 기능적인 측면에 초점을 맞춰 검증 및 테스트를 수행하기 때문에 모든 예외적인 상황에 대해 테스트하기 어렵다. 이에 본 논문에서는 서비스 조합의 강건성에 초점을 맞춰 이를

검증하는 방법을 제시한다. 본 논문의 방법은 조합된 웹 서비스에서 발생 가능한 모든 예외적인 상황을 수집하고 이를 발생시키는 가상의 환경을 구축함으로써 기술된 BPEL 프로세스가 각 상황에 대해 적절한 예외처리 방법을 적용하고 있는지 테스트 할 수 있다.

3. SOA 환경에서 서비스 조합 강건성 테스트 방법

3.1 서비스 조합의 강건성

서비스 조합(services composition)이란 복합 서비스를 개발하는 일련의 과정이라고 정의할 수 있으며, 하나의 서비스 내에서 프로세스 실행 흐름에 따라 외부의 다른 서비스를 호출하는 경우를 포함한다. 예를 들어, 그림 1은 동기식(synchronous) 웹 서비스를 조합한 여행 예약 서비스에 대한 BPEL 프로세스를 도식적으로 표현한 예이다. 이 프로세스에서 참여자는 크게 고객(client application)과 서비스 제공자(Flight, Hotel, Car services)로 구성되며, 서비스 제공자가 제공하는 서비스를 이용하여 조합된 여행 예약 서비스를 고객이 이용한다. 서비스 조합에 참여하는 각각의 서비스 제공자는 WSDL과 파트너 링크(PartnerLink)로 정의된다. 그림에서 오른쪽의 각 서비스 애플리케이션들은 BPEL 프로세스 내부에서는 파트너 링크 부분에 명시된다. 이때 기술되는 정보는 제공되는 서비스의 위치, 서비스 인터페이스의 이름, 입출력 데이터 등에 관한 정보들이다.

아래 BPEL 프로세스는 다음과 같이 동작한다. 일단 조합된 여행 예약 서비스의 사용자가 서비스를 요청하면 BPEL 실행엔진을 통해 BPEL 프로세스가 시작된다. 그림에서 <receive> 액티비티는 BPEL 프로세스 내에서 사용자의 요청을 받아들이는 부분이며, <assign> 액티비티는 사용자가 제공한 정보를 서비스 제공자의 메시지 타입에 맞춰 가공한다. 이후 <invoke> 액티비티에서는 Flight, Hotel, Car 서비스와 같이 실제 서비스를 호출하여 사용자에게 서비스를 제공한다. 예를 들어, 비행 예약 서비스를 통해 도착시간이 결정되면, 이를 기반으로 자동차 예약 시간을 결정할 수 있고, 이런 정보가 자동차 예약 서비스에 입력 정보로 전달된다. 이러한 일련의 서비스의 최종 수행 결과는 <reply> 액티비티를 통해 고객에게 전달된다.

이와 같이 조합된 새로운 서비스는 사용자로 하여금 각각의 서비스 제공 업체를 통해 별도로 예약할 필요 없이 하나의 복합 서비스를 통해 모든 예약 작업을 한 번의 입력으로 수행할 수 있게 해주어 시간과 노력을 절감할 수 있게 해준다. 그러나 이렇게 조합된 서비스가 강건한 서비스라 할 수 있을까? 조합된 여행 예약 서비스가 다양한 예외 상황에 대해 적절하게 대처할 수 있는 능력을 갖고 있는지 검증할 필요가 있다. SOA를 기반으로 한 서비스 조합의 특성상 분산된 다양한 서비스들이 조합되기 때문에 다양한 예외 상황이 발생할 수 있고 BPEL 프로세스는 이러한 상황에 적절하게 대처할

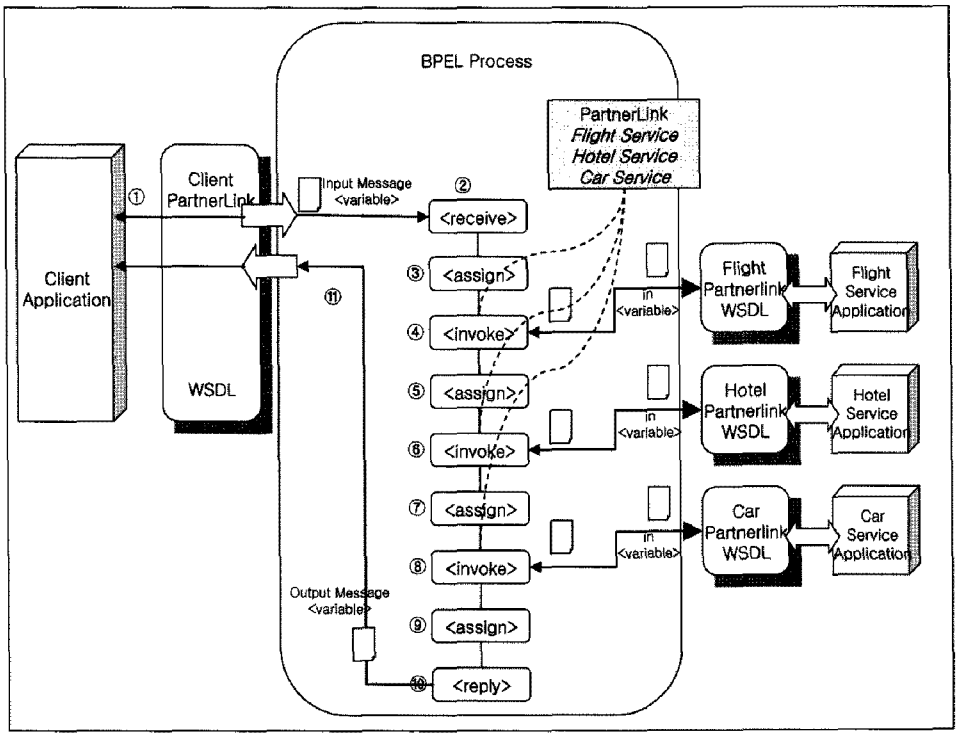


그림 1 여행 예약 서비스에 대한 서비스 조합

수 있는 능력을 갖고 있어야 한다. 그렇다면 어떤 서비스가 강건하다고 말할 수 있을까? 본 논문에서는 조합된 서비스가 다음과 같은 상황을 극복할 수 있어야 강건하다고 정의한다.

첫째, 개별 서비스의 에러(error)에 대해 대처할 수 있어야 한다. 예를 들어, 특정 상황에서 서비스 요청은 수락하였으되 비행기 예약 서비스가 내부적인 로직의 에러로 인해 더 이상 서비스를 제공할 수 없는 상황을 가정해보자. BPEL 프로세스는 서비스 제공자에게 서비스를 요청하고, 계속해서 응답을 기다릴 것이다. 혹은 일정 시간 이후에 서비스를 제공받을 수 없다는 것을 인지하고 전체 프로세스를 종료할 것이다. 이 둘의 경우 후자가 개별 서비스의 에러에 대해 더 잘 대처하는 프로세스 일 것이다. 그러나 BPEL 프로세스가 보다 더 강건한 서비스로 정의되기 위해서는 에러가 발생한 서비스를 대체할 수 있는 다른 서비스를 이용해 계속해서 서비스를 제공할 수 있어야 한다. 이를 위해서는 각 개별 서비스에서 발생할 수 있는 다양한 에러를 사전에 예측하고 그러한 에러에 적절하게 대응하기 위한 로직이 BPEL 프로세스 내에 포함되어야만 한다.

둘째, 개별 서비스 사용 중 발생할 수 있는 예외 사항(exception)을 처리할 수 있어야 한다. 이는 앞의 에러와는 다른 경우이다. 각 개별 서비스는 서비스 이용 중에 발생할 수 있는 다양한 예외사항에 대해 명세하고 있으며 조합된 서비스에도 이에 대한 처리 과정이 반드시 포함되어야 한다. 예를 들어, Hotel 서비스에서 사용자가 요구한 유형의 방이 없을 때 'NotAvailableRoom'이라는 메시지를 반환하는 것으로 예외처리를 수행한다고 가정하자. 조합 서비스인 여행 예약 서비스에서 이런 메시지가 발생할 경우 다른 호텔에서 같은 등급의 방을 찾거나, 같은 호텔에서 다른 등급의 방을 검색하여 예약할 수 있도록 기술할 수 있다. 따라서 보다 더 강건한 BPEL 프로세스를 구현하기 위해서는 서비스 조합에 참여하는 개별 서비스들의 예외 상황에 대한 정보를 수집하고, 각각의 예외 상황에 대한 처리 방법이 프로세스 내에 정의되어야 한다.

마지막으로 웹 서비스 수행 환경에서 발생하는 환경 오류(failure)를 처리할 수 있어야 한다. 조합 서비스는 기본적으로 네트워크 상에 분산되어 있는 다양한 웹 서비스를 통합하여 새로운 서비스를 구축한다. 따라서 BPEL 프로세스는 네트워크 상에서 발생할 수 있는 다양한 오류들에 대한 대처 능력을 갖고 있어야 한다. 예를 들어, 프로세스 진행 중 간헐적인 서버 연결 실패, 또는 연결 시간 초과와 같은 상황은 언제든지 발생 할 수 있다. 이러한 상황은 서비스 재시도와 같은 간단한 해결 방법을 프로세스에 기술해 줌으로써 해결할 수 있다. 또한 환경 오류는 각 개별 서비스를 구현하는 기술에 의해서도 발생할 수 있다. 예를 들어 SOAP의 구조상의 문제, 서비스의 접근 권한에 의해 발생하는 문제 등도 고려하여야 한다.

조합 서비스가 얼마나 강건한지 확인하기 위해서는 위와 같은 3가지 상황에 대한 대처 능력을 확인하는 작업이 필요하다. 다양한 예외적인 상황에 대해 적절한 대처 방법이 기술되어 있다면 이는 강건한 서비스 조합이라 말할 수 있다. 이러한 대처 방법에는 서비스 요청의 재시도, 다른 웹 서비스를 이용한 서비스의 재개, 더 이상 서비스를 제공할 수 없을 경우에 서비스 철회, 그리고 서비스 철회 시점 이전에 진행된 서비스에 대한 롤백(roll back)등의 방법이 있을 수 있다. 본 연구에서는 BPEL 기반의 서비스 조합의 강건성을 테스트하기 위해 다양한 유형의 오류 및 예외 상황을 발생시키는 가상의 환경을 구축하고, 다양한 시나리오로 테스트를 수행하는 자동화 방법을 제시한다.

3.2 웹 서비스 조합의 강건성 테스트 자동화 방법

앞서 언급한 것과 같이 BPEL 프로세스는 참여하는 각각의 서비스에서 발생 가능한 에러, 예외 상황, 그리고 다양한 환경에 의한 오류에 대한 대처 능력을 갖고 있어야 강건한 프로세스이다. 따라서 BPEL 프로세스의 강건성을 테스트하기 위해서는 다양한 예외 상황이나 오류를 발생시키고, 이에 대한 대처 능력을 확인해야 한다. 이를 위하여 이 논문에서는 BPEL 프로세스와 참여하는 웹 서비스들을 분석하여 예외 상황에 대한 정보를 자동으로 수집하고, 이를 기반으로 예외 상황이나 오류를 발생 시키는 가상의 환경을 자동으로 구축하는 방법을 제시한다. 또한 각각의 예외 상황에 대한 시나리오를 기반으로 테스트 케이스를 자동으로 생성함으로써 테스트 수행 과정을 자동화한다. 이러한 강건성 테스트를 자동화 하기 위하여 일반적인 소프트웨어 테스팅 절차에 따라 각 단계를 자동화 하는 방법을 제시한다. 이를 위해 다음과 같은 4단계의 작업을 수행한다.

Step 1. 테스트 계획: 이 단계에서는 테스트의 목적과 대상(target) 그리고 테스트의 범위를 지정한다(표 1). 계획 단계에서는 특별한 자동화 방법이 적용되지 않는다.

표 1 테스트 계획

테스트 목적	SOA 기반 서비스 조합의 강건성을 검증하기 위함
테스트 대상	SOA 기반 서비스 조합을 명시한 BPEL 프로세스
테스트 범위	SOA 기반 서비스 조합의 기능 및 강건성 테스트

Step 2. 테스트 준비

Step 2-1. 테스트 하니스(harness) 자동 생성: 이 단

계에서는 테스트 드라이버(driver)나 테스트 스텁(stub)과 같은 테스트 하니스를 준비한다. 이러한 테스트 하니스를 자동으로 생성하기 위해 BPEL 프로세스와 서비스 조합에 참여하는 서비스를 분석하여 필요한 정보를 추출하는 작업이 선행되어야 하며, 이를 기반으로 테스트 드라이버와 테스트 스텁을 자동으로 생성한다. 이때 다음과 같은 세부 작업이 수행된다.

1) BPEL 프로세스 분석
2) 참여 서비스 분석
3) 테스트 드라이버 자동 생성
4) 테스트 스텁 자동 생성

1) BPEL 프로세스의 분석: BPEL로 기술된 프로세스를 분석하여 다음과 같은 정보를 추출한다.

① 프로세스의 구조
② 참여하는 서비스의 목록
③ 프로세스 내의 예외처리 구조

그림 2는 WS-BPEL 2.0의 메타모델을 나타낸 것이다. 각각의 정보는 BPEL 프로세스를 구성하는 요소들을 분석하여 추출한다. 우선 ① 전체 프로세스의 구조를 분석하기 위해 <Process> 요소를 분석한다. WS-BPEL은 <Process>라는 최상위 요소를 시작으로 하나의 주요 <Activity>를 포함하며, 하나의 <Activity>는 다수의 <Activity>를 포함할 수 있다. 이러한 <Activity>에는 <invoke>, <assign>, <throw>, <exit>, <wait>, <sequence>, <if>, <while>, <repeatUntil>, <forEach>, <flow>, <scope> 등과 같은 종류가 있으며, 이런 요소들의 의미와 상호 연결 관계를 통해 전체 프로세스의 구조를 파악한다. 예를 들어, 그림 1의 여행 예약 서비스는 <assign>, <invoke> 액티비티를 이용하여 순차적인 프로세스를 정의한 것이다.

② 참여하는 서비스의 목록은 BPEL 프로세스에서 <PartnerLink> 요소를 분석하여 추출한다. <PartnerLink>는 비즈니스 프로세스 또는 다른 서비스에 대한 인터페이스를 기술하는데 사용하며, BPEL 프로세스는 파트너 링크에 저장된 정보를 이용하여 외부 서비스들에 접근할 수 있다. 파트너 링크는 WSDL의 포트타입(portTypes)을 이용하여 웹 서비스에 대한 인터페이스를 구성하는 작업 및 메시지 타입을 정의하게 되며 전송 프로토콜과 서비스의 위치를 간접적으로 정의한다. 예를 들어, 그림 1의 여행 예약 서비스의 Flight 서비스, Hotel 서비스, Car 서비스는 <PartnerLink>에 명시된다. 이 단계에서는 BPEL 프로세스 내에 정의된 참여 서비스의 목록만을 추출하며, 개별 서비스에 대한 구체적인 정보는 2)참여 서비스 분석 단계에서 추출한다. 그림 3은 BPEL 프로세스와 파트너링크 그리고 실제 서비스의 WSDL과의 관계를 도식화한 것이다. 그림에서 실제 서비스에 대한 정보는 WSDL 파일에 기술되어 있으며, 이러한 정보를 참조하여 <PartnerLink>를 정의한다. BPEL 프로세스는 이러한 <PartnerLink>정보를 이용하여 프로세스 내에 참여하는 서비스의 목록을 구성한다.

③ 프로세스 내의 예외처리 방법을 추출하기 위해서는 <faulthandler> 요소를 분석한다. BPEL 프로세스는

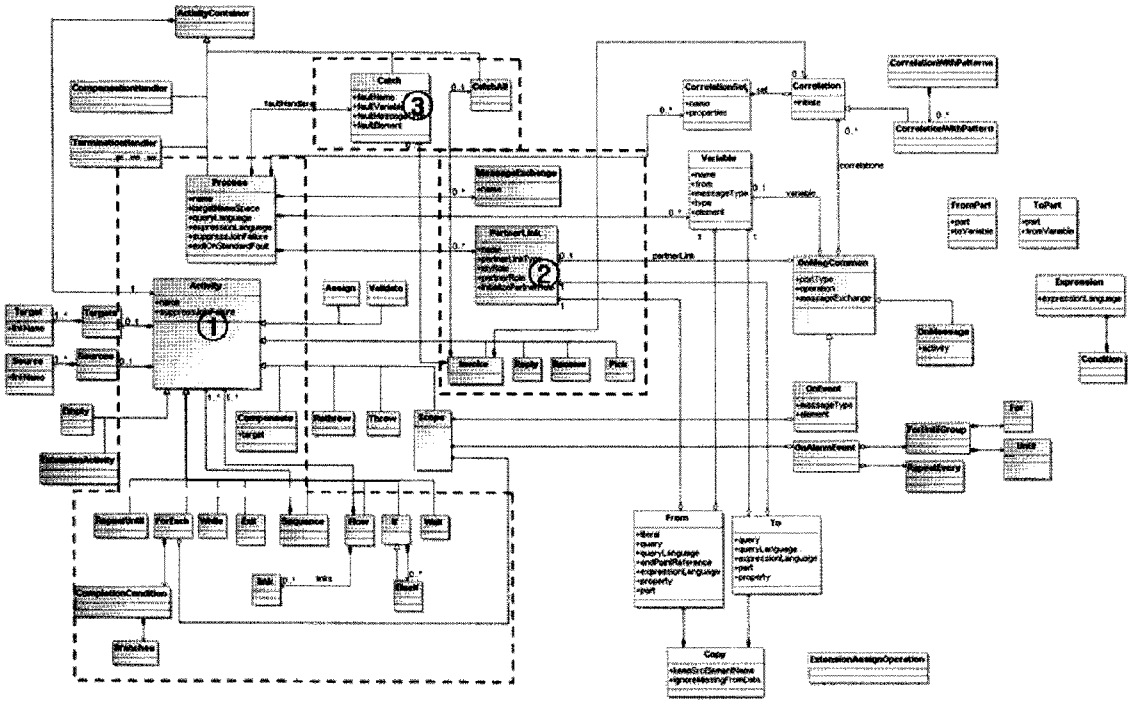


그림 2 WS-BPEL의 메타모델[12]

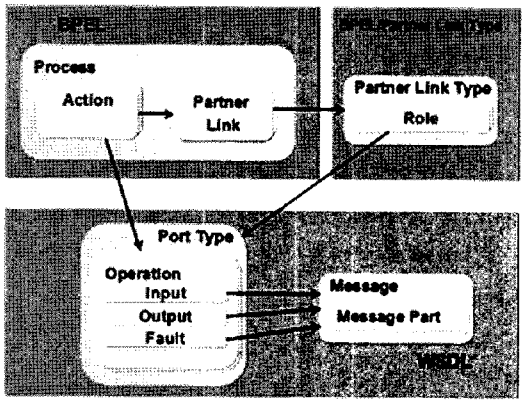


그림 3 BPEL 프로세스와 WSDL과의 관계

작업의 수행 중간에 발생하는 오류 상황을 처리하기 위한 방법으로 <faulthandler>요소를 이용한다. 이는 프로세스 처리 중 오류가 발생하는 경우 이를 캐치(catch)하여 다른 서비스를 호출하거나, 프로세스 수행을 종료하는 등의 작업을 할 수 있다. 예를 들어, 그림 1의 서비스 수행 중 비행기 표 예약이 끝난 시점에서 특정 호텔에 예약 가능한 방이 없다는 'NotAvailableRoom'의 예외 상황이 발생할 경우 <faulthandler> 요소에 다른 호텔을 통해 예약을 계속 하거나 전체 프로세스의 수행을 종료하는 것을 명시할 수 있다. 전체 프로세스를 종료할 경우 기존의 비행기 예약은 취소되어야 한다. 다음 예를 살펴보자. 그림 4는 호텔 예약 서비스에 대한 예외처리의 한 예를 보여준다.

```
<bpws:faultHandlers>
   <catch faultMessageType="ns1:errorMessage"
    faultName="lns:NotAvailableRoom" faultVariable="ServiceError">
    <reply partnerLink="client" portType="lns:HotelService" operation="reserveRoom"
      variable="ServiceError"
      faultName="lns:NotAvailableRoom"/>
   </catch>
  <catchAll>
    <empty/>
  </catchAll>
</bpws:faultHandlers>
```

그림 4 호텔 예약 서비스의 예외처리

그림 4의 예외처리 방법은 'NotAvailableRoom'의 예외 상황에 대해 아무런 처리 없이 오류 메시지를 반환한 후 프로세스를 종료하는 구조이다. 이 경우 앞서 수행된 비행기 예약 서비스의 수행 결과는 계속 유지되고 있기 때문에 결과적으로 문제가 될 수 있다. 따라서 프로세스를 종료할 경우 <faulthandler> 부분에 앞의 비행기 예약을 취소할 수 있는 내용이 정의되어야 할 것이다. 이처럼 BPEL의 예외처리 기능은 전체 프로세스의 강건성에 영향을 준다. 예외 상황에 대한 처리를 수행하지 않거나, 제대로 처리하지 않을 경우 프로세스 수행의 신뢰성을 저하시키는 요인이 되며, 네트워크 연결 실패와 같은 사소한 오류에도 전체 프로세스의 실패와 같은 극단적인 상황을 만들어 낼 수 있다. 본 논문에서는 이러한 예외처리 구조를 BPEL 프로세스로부터 추출하여 테스트 수행 이후에 결과를 판단할 때 활용하며, 향후 강건성 향상을 위해 개선할 부분으로 제시하고자 한다.

2) 참여 서비스 분석: BPEL 프로세스를 분석한 다음 참여 서비스의 목록을 기반으로 개별 서비스를 분석한다. 각 참여 서비스에 대한 구체적인 정보는 해당 서비스의 WSDL 파일에 기술되므로, 각각의 WSDL 파일을 분석하여 다음과 같은 정보를 추출한다. 이들 정보는 테스트 스텁의 생성에 활용된다.

① 서비스 이용에 필요한 데이터 타입 정보
② 서비스의 입/출력 메시지 정보
③ 서비스 인터페이스
④ 서비스의 예외처리 방법

또한 이 단계에서는 조합된 서비스의 WSDL 파일도 분석하여 위의 네 가지 정보를 추출한다. 조합된 서비스 역시 하나의 새로운 서비스로 표현되기 때문에 WSDL 파일을 갖고 있다. 조합 서비스의 WSDL 파일을 분석하면 테스트 드라이버를 생성하기 위해 필요한 정보를 추출할 수 있다. 그림 5는 WSDL의 메타모델을 나타낸다.

①의 <type> 요소는 서비스 호출 및 결과 반환을 위한 데이터 타입을 정의하는 부분이며, ②의 <message> 요소는 인터페이스의 입/출력 메시지 포맷을 명시하고 있다. ③ <interface>, <service>, <binding> 요소는 해당 서비스의 인터페이스와 접근 방법에 대한 명세이다. 그리고 ④ <fault> 요소는 서비스 실행 중 발생하는 오류를 명시하는 부분으로 오류가 발생하면 지정된 타입의 오류 메시지를 반환한다. 우리는 WSDL의 이러한 정보를 분석하여 BPEL 프로세스를 구성하는 참여 서비스에 대한 정보를 추출한다. 편의상 WSDL 파일은 크게 두 종류로 나눠진다. 하나가 조합 서비스를 명시하는 WSDL 파일로써 이는 그림 1에서 고객이 접근하는 조합 서비스인 여행 예약 서비스를 기술하는 WSDL 파일이고, 다른 하나는 개별 서비스를 명시하는 WSDL 파일로써 그림 1에서 비행기, 호텔, 자동차 예약 서비스를 기술하는 개별 서비스의 WSDL 파일이다. 그러나 사실 이러한 WSDL 파일들은 그 구조가 동일하고, XML로 기술되기 때문에 XML파서(parser)를 기반으로 한 분석 도구를 통해 위에 나열한 정보들을 쉽게 추출할 수 있다.

3) 테스트 드라이버 자동 생성: 이 논문에서 테스트 드라이버는 테스트를 수행할 때 고객의 역할을 수행하는 모듈이다. 테스트 작업을 수행하기 위하여 조합 서비스를 호출하고, 적절한 입력 데이터를 전달하는 역할을 수행한다. 이러한 테스트 드라이버는 앞서 설명한 조합 서비스의 WSDL을 분석하여 추출한 ①, ②, ③의 정보를 이용하여 생성한다.

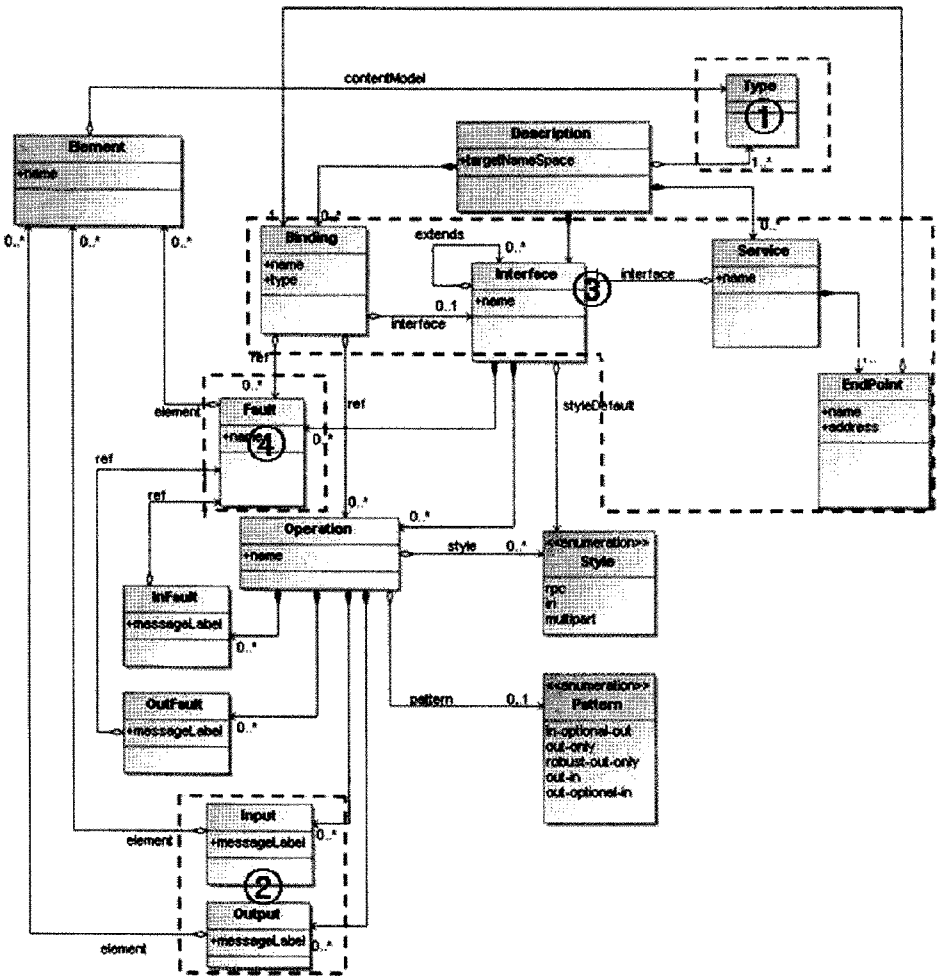


그림 5 WSDL의 메타모델[13]

4) 테스트 스텁(가상 서비스) 자동 생성: 이 논문에서는 서비스의 조합 환경을 고려하여 일반적인 테스트 스텁보다 더 복잡한 기능을 수행하는 테스트 스텁을 생성한다. 이러한 테스트 스텁은 서비스 조합에 참여하는 개별 서비스의 역할을 담당한다. 이러한 관점을 반영하여 이 논문에서는 테스트 스텁을 '가상 서비스(virtual service)'라 부른다. 소프트웨어 테스팅 분야에서 흔히 사용하는 테스트 스텁은 대체적으로 수동적이다. 즉 이런 테스트 스텁은 내부 로직을 갖지 않고 주어진 입력 파라미터와 무관하게 일정한 결과만을 반환한다. 그러나 이 논문에서의 가상 서비스는 약간의 내부 로직을 갖고 있으며, 주어진 입력 파라미터나 입력 제어에 따라 서로 다른 결과값을 반환한다. 이 논문의 목적이 BPEL 프로세스의 강건성을 테스트하는 것이기 때문에 생성된 가상 서비스들은 참여 서비스의 예외 상황이나 오류 등을 시뮬레이션 할 수 있게 여러 유형의 결과값이나 오류 등을 반환 하도록 설계되었다.

본 논문에서 BPEL의 강건성 테스트를 위해 가상 서비스를 사용하는 이유는 크게 세 가지이다. 첫째, 실제 서비스를 이용할 때 드는 비용 문제를 해결할 수 있다. 개별 서비스는 상용 서비스일 수도 있고, 이용하는데 비용이 부가될 수 있다. 다양한 테스트의 수행에 있어 이러한 비용은 엄청난 오버헤드가 될 수 있기 때문에 테스트 시점에서 실제 서비스를 이용하기는 어렵다. 둘째, 실제 서비스를 이용할 경우 다양한 오류가 발생할 가능성이 낮다. 어느 정도 안정된 서비스를 이용할 경우 테스트 수행 기간에 오류가 발생하지 않을 수 있다. 따라서 실제 서비스에서 어떠한 오류가 발생하기까지 기다리며 테스트하는 것은 매우 긴 시간을 요구할 수 있다. 이 논문의 가상 서비스는 실제 서비스의 기능을 완벽하게 수행하지는 않지만, 실제 서비스에서 발생 가능한 다양한 유형의 오류를 포함하고 있어 제약된 시간에 집중적으로 오류 및 예외 상황에 대한 대처 능력을 테스트할 수 있다. 마지막으로 BPEL 프로세스 개발 과정 중에 하향식(top-down) 방법을 이용하는 경우 실제 사용할 서비스가 결정되지 않은 상태에서 프로세스를 개발하기 때문에 테스트하기가 쉽지 않다. 따라서 가상 서비스는 이 방법을 지원하기에 적합한 요소라 할 수 있다.

가상 서비스 생성 단계에서는 BPEL 프로세스에서 조합될 실제 서비스를 대체할 가상 서비스를 생성하며, 가상 서비스는 다음과 같은 값을 반환한다.

- 정상 입력에 대한 정상 결과 반환
- 정상 입력에 대한 비정상 결과 반환
- 서비스에 명시된 예외 상황 반환
- 서비스에 명시되지 않은 예외 상황 반환

가상 서비스는 실제 서비스와 1:1로 대응되며, 앞의 2) 참여 서비스 분석 단계에서 추출한 인터페이스 정보와 예외처리 정보를 기반으로 생성된다. 가상 서비스는 단순히 참여 서비스 분석 단계에서 추출한 예외 상황뿐만 아니라 웹 서비스 동작 중 발생할 수 있는 일반적인 오류에 대해서도 다양한 예외 상황을 발생시킬 수 있다. 이를 위해 SOAP 오류, WSDL 오류, 네트워크 오류 등과 같은 일반적인 웹 서비스 오류들을 라이브러리 형태로 사전에 정의하고, 이를 이용해 테스트 수행 시 다양한 오류를 발생시킨다. 그림 6은 이 논문에서 제안하는 가상 서비스의 구조를 나타낸다.

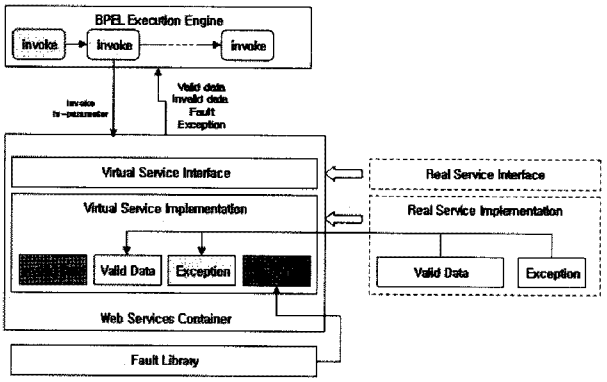


그림 6 가상 서비스 구조

또한 가상 서비스는 테스트 수행 과정에 관한 정보를 기록한다. 이는 테스트 수행 이후에 정상 결과를 반환했는지, 오류를 반환했는지에 대한 내용과 반환 값에 따른 BPEL 프로세스의 처리 과정을 비교하여 테스트 결과를 평가하기 위함이다. 그러나 개별 서비스에서 서비스 호출 및 반환 결과만을 기록하는 것은 의미가 없다. BPEL 프로세스 내에서 어떤 처리가 이루어졌는지 알 수 없기 때문이다. 따라서 BPEL 프로세스에서 프로세스 수행 중간에 각 액티비티의 처리 과정을 기록할 수 있도록 프로세스를 확장하는 것이 필요하다. 이에 본 논문에서는 각 액티비티의 수행 이후에 수행 내역을 기록할 수 있도록 기록 액티비티를 추가하여 BPEL 프로세스의 전체 수행 내역을 기록한다.

Step 2-2. 테스트 케이스의 준비

1) 테스트 시나리오의 준비

이 논문에서 테스트 케이스는 BPEL 프로세스를 동작시키는 시나리오로 표현된다. 즉 조합 서비스를 구성하는 참여 서비스들을 프로세스 로직에 따라 호출하는 일련의 서비스 호출 순서로 표현된다. 시나리오에는 데이터도 함께 표현되는데 이러한 데이터는 입력 데이터와 제어 데이터로 구분된다. 입력 데이터는 조합 서비스를 호출할 때 제공하는 입력 값이며, 제어 데이터는 가상 서비스를 구동시키기 위한 데이터이다. 제어 데이터에 의해 각 가상 서비스는 앞서 설명한 네 가지 종류의 결과값 중 하나를 반환한다. 테스트 시나리오의 구성은 다음과 같이 순서쌍들의 순서로 표현된다.

테스트 시나리오의 구조: [<S_{com}, {input values}, con>, <S_1, con>, <S_2, con>, ···]

여기서 첫 번째 순서쌍은 조합 서비스의 호출(S_{com})을 의미하며, 입력 값들과 제어 데이터를 갖는다. 이를 제외한 나머지 순서쌍들은 개별 서비스 호출(S_i)과 이에 필요한 제어 데이터를 갖는다. 예를 들어, 그림 1의 여행 예약 서비스에 대하여 표 2와 같은 테스트 시나리오를 생성할 수 있다.

입력 데이터와 제어 데이터의 조합을 통해 많은 수의

표 2 테스트 케이스

Test scenario 1	Test scenario 2	Test scenario 3	..
< S_{com}, {f_date:032209~032809; f_dest:London; h_date:032209~032809; h_city:London; h_grade:4star; h_room:twin; c_date:032209~032809; c_type:sedan}, **valid**>	< S_{com}, {f_date:041209~041809; f_dest:Seattle; h_date:041209~041809; h_city:Seattle; h_grade:4star; h_room:single; c_date:041209~041809; c_type:sedan}, **valid**>	< S_{com}, {f_date:032209~032809; f_dest:Chicago; h_date:032209~032809; h_city:Chicago; h_grade:4star; h_room:twin; c_date:032209~032809; c_type:sedan}, **valid**>	
< S_{Flight}, **valid**>	< S_{Flight}, **valid**>	< S_{Flight}, **invalid**>	
< S_{Hotel}, **valid**>	< S_{Hotel}, **exception**>	< S_{Hotel}, **fault**>	
< S_{Car}, **valid**>	< S_{Car}, **exception**>	< S_{Car}, **exception**>	

테스트 케이스를 자동으로 생성할 수 있다. 생성되는 테스트 케이스의 총 개수는 다음과 같다. 여기에서 $n(S_x,y)$는 타겟 서비스 Sx에 적용되는 입력 데이터 y에 대한 동치 클래스(equivalent class)의 개수를 나타내며, k는 참여 서비스의 개수를 나타낸다.

테스트 케이스의 개수 = $n(S_{com}, input\_data) * n(S_{com}, control) * \Pi_{i=1}^{k} n(S_i, control)$;

예를 들어, 그림 1에서 조합 서비스인 여행 예약 서비스가 입력 데이터에 대해 32개의 동치 클래스, 제어 데이터에 대해 4개의 동치 클래스를 갖고 있고, Flight 서비스, Hotel 서비스, Car 서비스 각각이 제어 데이터에 대해 4개의 동치 클래스를 갖고 있다면 생성되는 테스트 케이스의 전체 개수는 32*4*4*4*4인 8,192개이다.

2) 테스트 충분성(adequacy) 평가

이 논문에서는 참여 서비스에서 발생 가능한 모든 예외적인 상황에 대해 BPEL 프로세스가 강건한지 테스트할 수 있도록 테스트 케이스가 작성된다. 따라서 어떤 테스트 시나리오가 수행되면 참여 서비스의 조건들 중 하나는 실행(커버) 된다. 예를 들어, 여행 예약 서비스의 테스트에서 어떤 시나리오를 수행하였더니 호텔 서비스는 정상적인 결과 값을 반환하였고, 렌터카 서비스는 'SoldOut'과 같은 예외 상황을 반환하였다면 이 시나리오의 수행을 통해 호텔 서비스의 정상 조건과 렌터카 서비스의 예외 상황에 대해 BPEL 프로세스가 테스트된 것이다. 다시 말해 호텔 서비스의 정상 조건과 렌터카 서비스의 예외 상황 조건이 커버된 것이다. 참여하는 모든 서비스들은 반환 값에 따라 정상, 비정상, 예외 및 오류의 네 가지 조건들을 가진다. 이러한 커버리지 기준(coverage criteria)을 통해 테스트 충분성을 평가할 수 있다. 이 논문에서는 테스트 충분성의 평가 기준을 다음과 같이 정의한다.

[테스트 충분성 평가 기준] 어떤 참여 서비스의 모든 조건들이 커버된다면 그 서비스는 커버되었다고 정의한다. 계속해서 모든 참여 서비스가 커버된다면 테스트는 충분하다고 평가한다.

이러한 테스트 충분성 기준은 테스트 케이스를 추출하는 데에도 활용할 수 있다. 즉 각 서비스의 모든 조건들을 커버할 수 있도록 테스트 케이스를 선별하면 된다. 테스트 케이스를 생성하기 위한 방법으로는 각 서비스의 조건들을 조합하여 생성하는 방법과 특정 서비스의 조건들은 값을 변경하되 다른 서비스의 조건들은 값을 고정하여 생성하는 방법이 있다. 후자의 경우 최소한의 테스트 케이스 집합을 구축할 수 있다.

Step 3. 테스트 수행의 자동화: 테스트 수행 단계에서는 세 가지 작업이 수행된다. 첫 번째 작업은 가상 서비스를 배치하는 작업이다. 이는 Step 2에서 생성된 가상 서비스를 로컬 서버에 배치하는 과정을 의미한다. 이를 위해 가상 서비스에 대한 WSDL 파일을 생성하고, BPEL 프로세스의 로직에서 외부 서비스에 대한 참조를 모두 로컬 서비스 참조로 변경한다.

두 번째 작업은 테스트 수행이다. 이는 배치된 가상 서비스를 이용하여 BPEL 프로세스를 실행시키는 단계이다. 이를 위해서 클라이언트 애플리케이션을 대신할 테스트 드라이버와 테스트 케이스를 이용 한다. 테스트 드라이버가 BPEL 프로세스로 표현된 조합 서비스를 호출하면 BPEL 수행 엔진에 의해 BPEL 프로세스가 수행된다. 이 때 BPEL 프로세스는 실제 서비스가 아닌 가상 서비스를 호출하도록 변경되었기 때문에 로컬 서버의 가상 서비스를 호출한다. 이러한 조합 서비스의 호출은 앞에서 생성한 모든 테스트 케이스의 수만큼 반복된다. 그림 7은 가상 서비스를 이용한 BPEL 프로세스의 강건성 테스트 수행 과정을 표현한 것이다. 테스트 수행 과정 중에 테스트 시나리오에 기술된 다양한 예외, 오류, 또는 정상적인 상황이 발생하고 이에 대한 BPEL 프로세스의 대처 능력이 검증된다. 그리고 각각의 테스트 케이스에 대한 수행 결과는 로그 파일에 기록된다.

세 번째 작업은 테스트 작업이 수행되고 난 다음 모든 것을 테스트 이전의 상태로 회복시키는 작업이다. 이 과정에서는 로컬 서버에 배치되어 있는 가상 서비스를 삭제하고 BPEL 프로세스를 원상태로 복구하는 작업(로컬 서비스 참조를 외부 서비스 참조로 변경)이 수행된다.

Step 4. 테스트 평가: 테스트 수행이 완료되면 테스트 평가 작업을 수행한다. 이 과정은 테스트 수행 기록을 통해 BPEL 프로세스가 개별 서비스에서 발생한 비정상적인 상황에 대해 적절하게 대처하였는지 여부를 통해 판단한다. 예를 들어, 여행 예약 서비스에서 호텔 서비스가 'NotAvailableRoom'과 같은 예외 상황을 반환하였을 경우 오류 메시지 반환 후 프로세스를 종료하거나, 동일 호텔에서 다른 등급의 방을 검색하여 반환하거나, 다른 호텔에서 동일 등급의 방을 검색하여 반환할 수 있다. 프로세스 종료의 경우 이미 수행된 다른 서비스들에 대해 그 상태를 유지하거나 혹은 롤백을 통해 취소할 수도 있다. 이와 같이 하나의 예외 상황에 대해 다양한 예외 처리가 수행될 수 있는데 그 중에서 어떤 예외 처리가 프로세스 작성자의 의도와 부합하는지 판단해야 정확한 테스트 평가가 이루어질 수 있다. 즉 각 시나리오의 수행 결과가 프로세스 작성자의 의도에 부합하면 테스트의 성공(Pass)으로, 그렇지 않다면 실패(Fail)로 판단한다. 실패의 경우 BPEL 프로세스가 단순하게 실행을 멈추었다거나, 무의미하게 프로세스를 종료하는 경우, 또는 적절하지 못한 메시지를 반환하는 경우이며 이러한 것이 BPEL 프로세스의 강건성을 저해하는

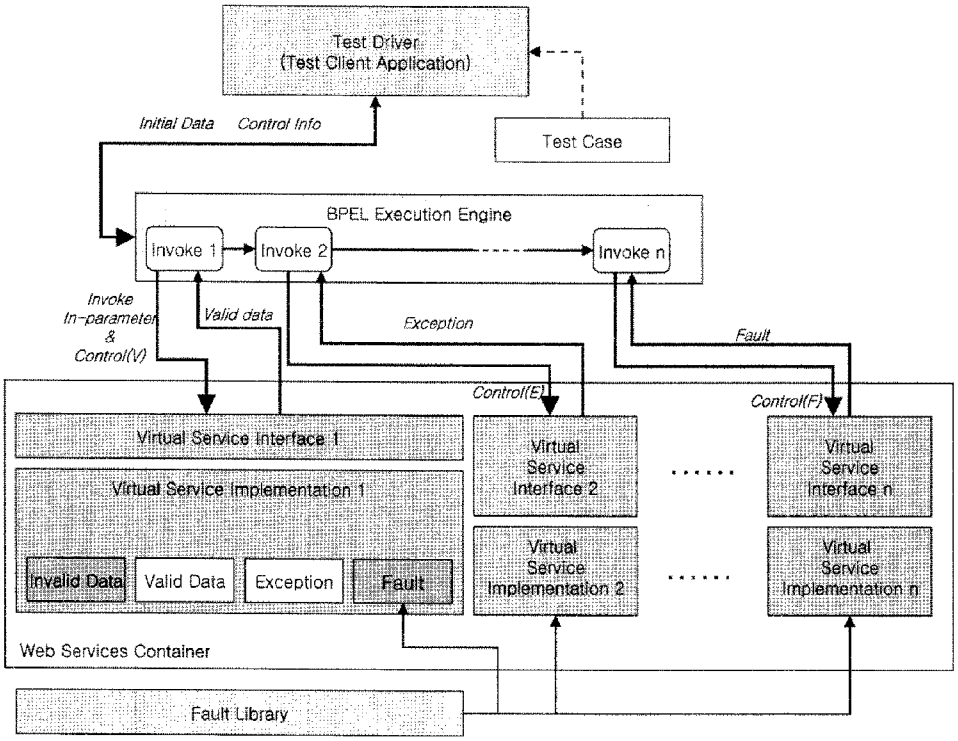


그림 7 테스트 수행

요인이 된다. 또 한가지 중요한 것은 예외적인 상황에서 프로세스가 완료되더라도 무조건 테스트 성공으로 볼 수 없다. 이는 예외처리 방법에 따라 프로세스가 완료된 것인지, 아니면 적절하지 못한 방법으로 완료된 것인지 확인해 볼 필요가 있다. 현재의 버전에서는 BPEL 프로세스 작성자의 의도를 표현할 수 있는 수단을 제공하고 있지 않다. 따라서 테스트 수행 기록을 바탕으로 사용자가 각 서비스의 비정상적 상황에 대해 BPEL 프로세스가 적절한 예외처리를 수행하였는지를 판단하여야 한다. 향후에 프로세스 작성자의 의도를 간단하게 표현할 수 있는 수단을 제공할 계획이다.

4. 테스트 자동화 도구의 구조

그림 8은 서비스 조합의 강건성을 테스트하기 위한 테스트 도구의 아키텍처를 보여준다. 3 절에서 설명한 네 단계의 테스트 과정을 지원하기 위해 BPEL 분석기, WSDL 분석기, 가상 서비스 생성기, 배치 관리자 등이 있으며, 테스트 환경을 생성하고 테스트 수행을 자동화하기 위해 테스트 드라이버 생성기가 존재한다. 또한 테스트 수행 과정을 기록하기 위한 로깅 모듈과 기록된 정보를 통해 테스트 결과를 평가하기 위한 테스트 결과 분석기가 존재한다.

BPEL 분석기는 테스트 대상이 되는 BPEL 프로세스를 분석하여, 전체 프로세스의 구조, 참여 파트너 서비스 정보 및 예외처리 구조를 파악한다. WSDL 분석기는 각 파트너 서비스가 제공하는 WSDL 파일을 분석하여 인터페이스, 입/출력 데이터 타입, 메시지 포맷, 예외처리 방법 등을 파악한다. 원활한 테스트 수행을 위해 실제 클라이언트 역할을 수행할 테스트 드라이버를 생성하는데, 이것은 BPEL 프로세스로 기술된 조합 서비스의 WSDL 파일을 분석하여 생성한다. 정보 추출기는 분석된 각각의 WSDL과 BPEL 프로세스로부터 테스트 환경 및 수행에 필요한 정보를 추출하여 내부 데이터로 저장한다. 필요한 정보가 모두 추출되면 테스트 드라이버가 생성되고, 서비스 조합에 참여하는 개별 서비스에 해당하는 가상 서비스를 생성한다. 생성된 가상 서비스는 배치 관리자에 의해 로컬 서버의 웹 서비스 컨테이너에 배치됨으로써 테스트 환경이 구축된다. 테스트 수행은 BPEL 실행 엔진에 의해 수행된다. 사용자는 테스트 드라이버를 구동시켜 조합 서비스를 호출하게 한다. 조합 서비스가 호출되면 그것은 BPEL 실행 엔진에 의해 수행된다. BPEL 프로세스 수행 중 참여 서비스 호출 시점에 가상 서비스가 호출된다. 이 때 서비스의 수행 과정 및 처리 결과는 로깅 모듈을 통해 기록된다. 테스트 수행이 완료되면, 테스트 결과 분석기를 통해 다양한 예외 상황에 대한 조합 서비스의 대처 능력을 분석한다.

5. 사례 연구

이 절에서는 논문에서 제시하는 테스트 프레임워크를 적용한 사례 연구에 대해 소개한다. 이 사례는 현재 한

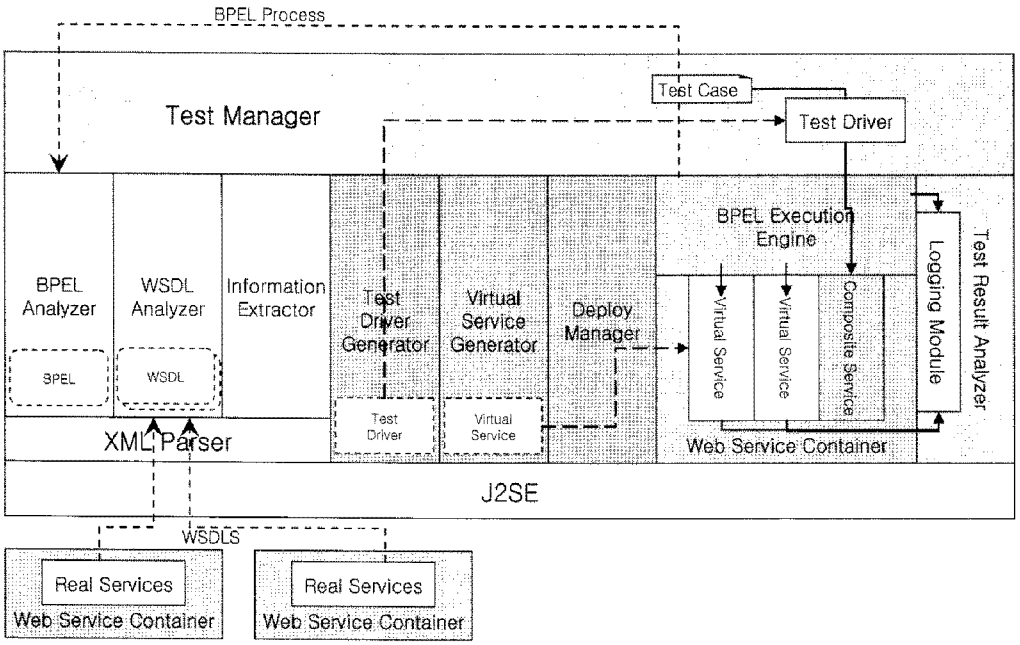


그림 8 테스트 자동화 도구 아키텍처

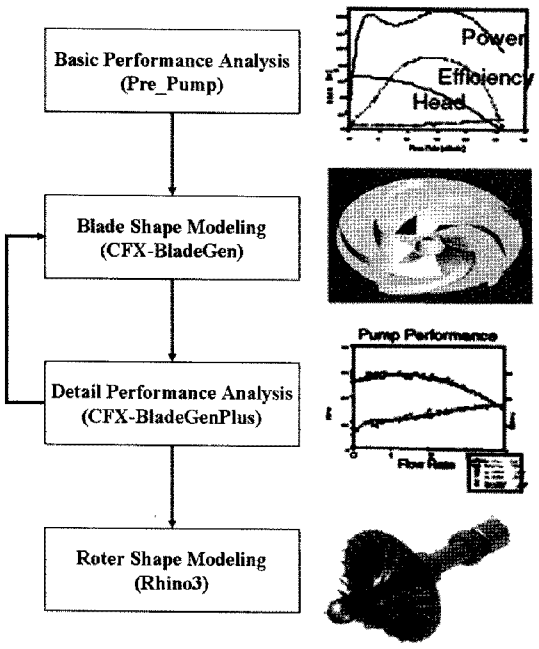
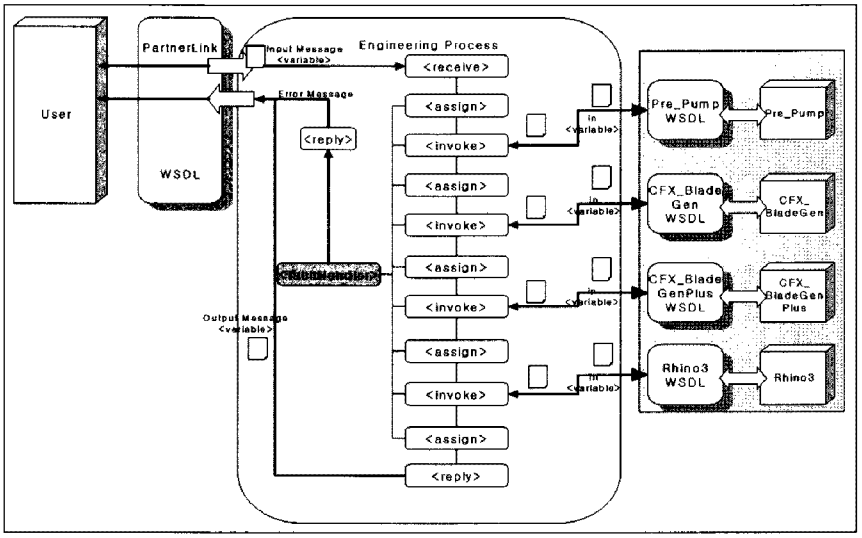


그림 9 펌프 설계 애플리케이션의 프로세스

국기계연구원의 웹 서비스 기반 e-엔지니어링 프레임워크에서 동작하는 펌프 설계 애플리케이션에 적용한 것이다. 그림 9는 펌프 설계를 위한 엔지니어링 프로세스를 도식화한 것이다. 이 프로세스는 4개의 하위 태스크로 이루어져 있으며, 각 태스크는 분산된 엔지니어링 소프트웨어에 의해 수행된다. 그림에서 사각형은 수행될 태스크를 표현하며, 그 태스크를 수행할 소프트웨어의 이름이 괄호 안에 표시되어 있다. e-엔지니어링 프레임워크를 개발하는 과정에서 프로세스의 신뢰성 향상을 위해 개선된 엔지니어링 프로세스가 얼마나 강건해 졌는지 평가하기 위해 강건성 테스트 방법을 적용하였다.

e-엔지니어링 프레임워크는 웹 서비스를 기반으로 분산된 엔지니어링 소프트웨어를 통합하기 위한 기반을 제공하고 있으며, BPEL을 이용하여 엔지니어링 프로세스를 모델링 하였다. 보다 자세한 사항은 논문 [14]를 참고하기 바란다. 그림 10은 위 엔지니어링 프로세스에 대한 BPEL 프로세스의 일부분과 이 BPEL 프로세스를 추상화한 그림이다.

이 펌프 설계 애플리케이션은 현재 한국기계연구원에서 다양한 펌프를 설계하는데 널리 사용하고 있다. 그러나 그림 10의 기존 펌프 설계 프로세스는 몇몇 예외 사항을 제외하고는 나머지 예외 상황이나 오류에 대해 적절한 예외 처리 방법을 제공하고 있지 않았다. 특히 하나의 태스크가 실패할 경우 대부분 전체 프로세스의 실패로 간주하기도 하였다. 따라서 어느 정도의 오류가 발생할 경우 프로세스를 종료하였기 때문에 작업의 효율이 낮았다. 이를 개선하기 위해 우리는 엔지니어링 프로세스에서 발생 가능한 예외 상황을 파악하고 각 상황에 맞는 예외 처리 기법을 제시하였다. 논문 [15]에서 e-엔지니어링 프레임워크에서 프로세스의 신뢰성을 향상시키기 위해 엔지니어링 프로세스를 확장하는 방법을 제시하였다. 구체적으로 Retry, Alternate, Continue, Confirm, Stop & Notify(S&N)와 같은 요소들을 엔지니어링 프로세스에 도입하였고, 그로 인해 다양한 예외 상황에 대해 프로세스가 그 수행을 멈추지 않고 계속해서 목적했던 서비스를 제공하게 함으로써 서비스의 신뢰성을 향상시켰다.




```
<?xml version="1.0"?>
<bpws:process xmlns:...../">
  <bpws:import location="../PrePumpService.wsdl"
    namespace="http://192.168.0.2/Services/PrePumpService.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/">
  </bpws:import>
  ......
  ......
  <bpws:partnerLinks>
    <bpws:partnerLink  partnerRole="prePumpService" name="prepump"
      partnerLinkType="ns:prePumpLinkType">
    </bpws:partnerLink>
  ......
  </bpws:partnerLinks>
  ......
  <bpws:faultHandlers>
    <catchAll>
      <reply partnerLink="client" portType="lns:pumpDesignService"
        operation="doPumpDesignService"
        variable="errorMessage"
        faultName="lns:cannotHandleRequest"/>
    </catchAll>
  </bpws:faultHandlers>
  <bpws:eventHandlers/>
    <bpws:sequence>
    <bpws:receive partnerLink="customer" portType="ns:pumpDesignServicePT"
      operation="request" variable="pumpDesignInputMessage" createInstance="no">
    </bpws:receive>
    <bpws:invoke partnerLink="prepump" portType="ns:prePumpPT" operation="doPrePumpService"
      inputVariable="prePumpInputMessage" outputVariable="prePumpOutputMessage">
    </bpws:invoke>
    ......
    <bpws:reply partnerLink="client" portType="ns:pumpDesignPT" operation="request"
      variable="completeMessage">
    </bpws:reply>
  </bpws:sequence>
</bpws:process>
```

① ② ③ ④ ⑤

그림 10 펌프 설계에 대한 BPEL 프로세스

펌프 설계 애플리케이션의 두 개의 버전을 대상으로 강건성 테스트를 수행하였다. 하나는 기본적인 몇 가지 예외 사항만 처리하는 기존의 펌프 설계 프로세스이고, 다른 하나는 논문 [15]에서 제안한 예외 처리 요소들을 포함하여 예외 처리 기능이 강화된 펌프 설계 프로세스이다. 테스트 작업은 그림 8의 도구에서 수행되었으며, 각 엔지니어링 태스크에 대응되는 가상 서비스를 구축하였다. 그림 11은 엔지니어링 태스크들 중 'Basic Performance Analysis' 서비스에 대한 가상 서비스인 Pre_Pump 서비스의 구조를 보여준다.

그림에서 볼 수 있듯이 가상 서비스는 실제 서비스의 인터페이스인 doPrePumpService()라는 인터페이스를 구현하고 있다. 내부적으로는 사전에 정의해놓은 정상 데이터 파일들(valid data file)과 비정상 데이터 파일들(invalid data file)을 저장하고 있다. 정상 데이터 파일은 실제 Pre_Pump 소프트웨어가 정상적으로 동작할 때 반환되는 데이터 파일을 저장한 것이며, 비정상 데이터 파일은 정상적으로 반환되는 파일의 내용을 인위적으로 변형한 데이터 파일을 저장한 것이다. 서비스 수행의 결과값을 반환할 때, 정상 수행의 경우 성공적인 수행을 의미하는 code 0과 함께 정상 데이터 파일을 반환하고, 서비스 내부 로직의 에러 상황을 모사하고자 하는 경우 code 0과 함께 비정상 데이터 파일을 반환한다. 여기서 code 0을 반환하는 이유는 특별한 예외 사항이나 환경에 의한 오류가 발생하지 않았기 때문이다. 가상 서비스는 실제 서비스에서 발생 가능한 예외 상황이나 오류도 반환하는데 실제 서비스에서 발생 가능한 예외 상황은 WSDL을 통해 파악할 수 있으며, Pre_Pump 소프트웨어의 경우 7개의 예외 상황이 정의되어 있다. 오류의 경우 웹 서비스 환경에서 발생 가능한 오류를 반환하도록 구현하였으며, SOAP, WSDL, 네트워크 등의 오류를 저장하고 있는 오류 라이브러리에서 27개의 오류를 활용하였다. 표 3은 사례 연구의 Pre_Pump 가상 서비스의 수행 결과의 상태를 나타내는 상태 코드로서 수행

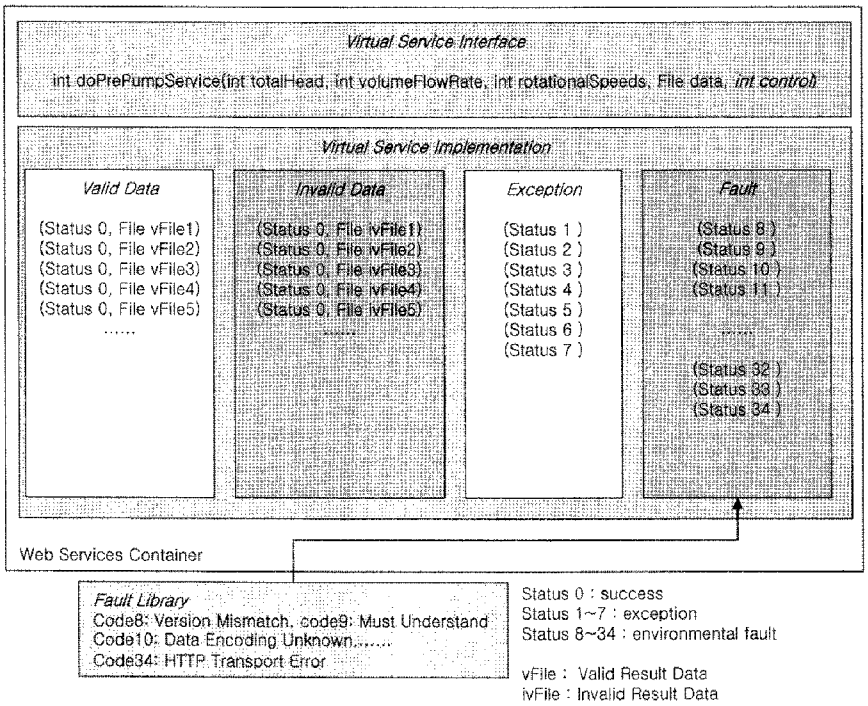


그림 11 Pre_Pump 가상 서비스의 구조

성공과 예외 상황, 환경 오류로 분류되어 있다.

테스트 케이스는 에러, 예외 사항, 오류 등의 비정상적인 결과를 반환할 수 있도록 생성하였다. 앞서 3절에서의 테스트 케이스는 여러 서비스에서 발생 가능한 예외 사항이나 오류를 조합하여 생성한 것이지만, 사례 연구에서는 각 서비스 별로 하나의 비정상적 결과에 대하여 테스트 할 수 있도록 테스트 케이스를 준비하였다. 표 4는 펌프 설계 프로세스에 대한 테스트 케이스의 예를 보여준다. 테스트 케이스는 Pre_Pump 소프트웨어 사용 중에 발생할 수 있는 모든 비정상적 상황에 대해 적어도 한번씩 테스트 할 수 있도록 작성되었다.

표 4에서 세 번째 열은 기존 펌프 설계 프로세스에 대한 테스트 수행 결과이고, 마지막 열은 예외 처리 기능이 강화된 펌프 설계 프로세스에 대한 테스트 수행 결과를 나타낸다. 결과에서 'Pass'는 BPEL 프로세스 내에서 해당 상황에 대해 적절하게 처리함으로써 BPEL 프로세스가 성공적으로 수행되었음을 의미한다. 'Fail'은 해당 상황에 대해 BPEL 프로세스의 수행이 실패하였음을 의미한다. 그리고 '--'는 Pass나 Fail이 즉시 결정되지 않는 경우에 해당한다. 예를 들어, 테스트 케이스 3번과 같이 사용자의 개입이 필요할 때 사용자의 개입 상황에 따라 Pass나 Fail이 결정되는 경우를 의미한다. 테스트 수행 결과를 통해 기존 펌프 설계 프로세스는 몇몇 예외 사항만 처리할 뿐이지 그 외의 예외 사항이나 웹 서비스 환경에 의한 오류는 전혀 고려하고 있지 않음을 알 수 있었다. 그에 반해 예외 처리 기능이 강화된 버전의 프로세스는 모든 예외 사항에 대해 적절하게 처리할 뿐만 아니라 환경 오류에 대해서도 적절하게 대응하고 있음을 알 수 있었다.

예외 처리 기능이 강화된 펌프 설계 프로세스는 BPEL 프로세스 내에서 서비스를 호출 한 후 반환되는 결과 값에 따라 적절한 예외 처리를 수행할 수 있도록 BPEL 프로세스가 확장된 버전이다. 예를 들어, 프로세스에서 <invoke>후에 상태 코드 code 1이 반환되었다면 입력 속성값이 잘못된 것이기 때문에 이를 변경하여 다시 서비스를 호출하도록 프로세스가 수정되었다. 또한 경우에 따라서는 프로세스 내의 예외 처리 로직만으로 예외 사항을 처리하기 어려울 경우 사용자의 개입이 필요하며 이를 위해 특정 예외 사항이나 오류가 발생하면 이를 사용자에게 통보하도록 확장하였다. 그림 12는 Pre_Pump 서비스에 대해 예외처리 기능을 확장한 BPEL 프로세스의 형태를 보여준다. 그림 10과 비교할 때 Pre_Pump 서비스 호출 다음에, 즉 <invoke> 요소 다음에, <faultHandler> 요소가 위치하고 있는 것이 다른 점이다.

기존 펌프 설계 애플리케이션은 대부분의 비정상적 상황에 대해 프로세스를 중단하였지만, 그림 12와 같이 예외 처리 기능을 강화한 새로운 펌프 설계 애플리케이션은 심각한 오류 상황을 제외하고는 프로세스의 수행을 중단하지 않기 때문에 서비스 수행의 성공 확률도 증가하였다. 결국 새로운 펌프 설계 애플리케이션은 보다 더 강건한 서비스 조합이라 할 수 있다.

표 3 Pre_Pump 가상 서비스의 수행 결과의 상태 코드

Status Code	Error Type	Error Message	Description
Code 0	--	--	Pre_Pump 소프트웨어가 정상 결과 반환
Code 1	Exception	Invalid Input Property	Pre_Pump 소프트웨어에 대한 입력 속성값이 잘못된 경우
Code 2	Exception	Invalid Input Data	Pre_Pump 소프트웨어에 대한 입력 데이터 파일이 잘못된 경우
Code 3	Exception	Variable Missing	Pre_Pump 소프트웨어의 수행에 필요한 입력이 없는 경우
Code 4	Exception	Job Queue Is Full	Pre_Pump 소프트웨어에 대한 작업 요청의 최대수를 초과한 경우
Code 5	Exception	Check Variable Value	Pre_Pump 소프트웨어에 대한 입력 값이 잘못된 경우
Code 6	Exception	Application Version Mismatch	Pre_Pump 소프트웨어의 버전이 일치하지 않은 경우
Code 7	Exception	Something Missing	Pre_Pump 소프트웨어의 수행 중 필요한 정보를 찾을 수 없는 경우
Code 8	Environment	Version Mismatch	SOAP 메시지 버전이 일치하지 않은 경우
Code 9	Environment	Must Understand	SOAP 메시지 헤더 부분에 필요한 정보가 누락된 경우
Code 10	Environment	Data Encoding Unknown	SOAP 메시지 내용의 인코딩 방식에 대한 정보가 일치하지 않는 경우
Code 11	Environment	Sender	SOAP 메시지 송신자에 문제가 발생한 경우; 예를 들어, 인증되지 않은 송신자의 메시지
Code 12	Environment	Receiver	SOAP 메시지 수신자에 문제가 발생한 경우; 예를 들어, 수신된 메시지가 변조되어 수신자에서 처리하지 못하는 경우
.....	.....	.....	.....
Code 30	Environment	WSRR Mapping Fault	WSDL과 UDDI의 맵핑이 정확하게 이루어지지 않은 경우
Code 31	Environment	Invalid Security	서비스에서 요구하는 보안 정책을 만족하지 못하는 경우
Code 32	Environment	Failed Authentication	고객 인증이 실패한 경우
Code 33	Environment	Cannot Access WSDL	서비스의 WSDL에 접근할 수 없는 경우
Code 34	Environment	HTTP Transport Error	메시지를 송신할 수 없는 경우; 예를 들어, 네트워크 장비에 권한이 없어 정상적으로 메시지를 송신할 수 없는 경우

표 4 Pre_Pump 서비스에 대한 테스트 케이스 및 테스트 결과

Test Case	Test Scenarios	Test Result(Old)	Test Result(New)
1	<S_{com},{},valid><S_1,**valid**><S_2,valid><S_3,valid><S_4,valid>	Pass	Pass
2	<S_{com},{},valid><S_1,**code1**><S_2,valid><S_3,valid><S_4,valid>	Fail	Pass
3	<S_{com},{},valid><S_1,**code2**><S_2,valid><S_3,valid><S_4,valid>	Fail	-- (Confirm)
4	<S_{com},{},valid><S_1,**code3**><S_2,valid><S_3,valid><S_4,valid>	Fail	-- (S&N)
5	<S_{com},{},valid><S_1,**code4**><S_2,valid><S_3,valid><S_4,valid>	Pass	Pass
6	<S_{com},{},valid><S_1,**code5**><S_2,valid><S_3,valid><S_4,valid>	Fail	Pass
7	<S_{com},{},valid><S_1,**code6**><S_2,valid><S_3,valid><S_4,valid>	Pass	Pass
.....	.....	.....	.....
34	<S_{com},{},valid><S_1,code33><S2,valid><S3,valid><S4,valid>	Fail	Pass
35	<S_{com},{},valid><S_1,code34><S2,valid><S3,valid><S4,valid>	Fail	Pass

S_{com}: composite service; S_1: S_{pre_pump}; S_2: $S_{bladegen}$; S_3: $S_{bladegenplus}$; S_4: S_{rhino3}

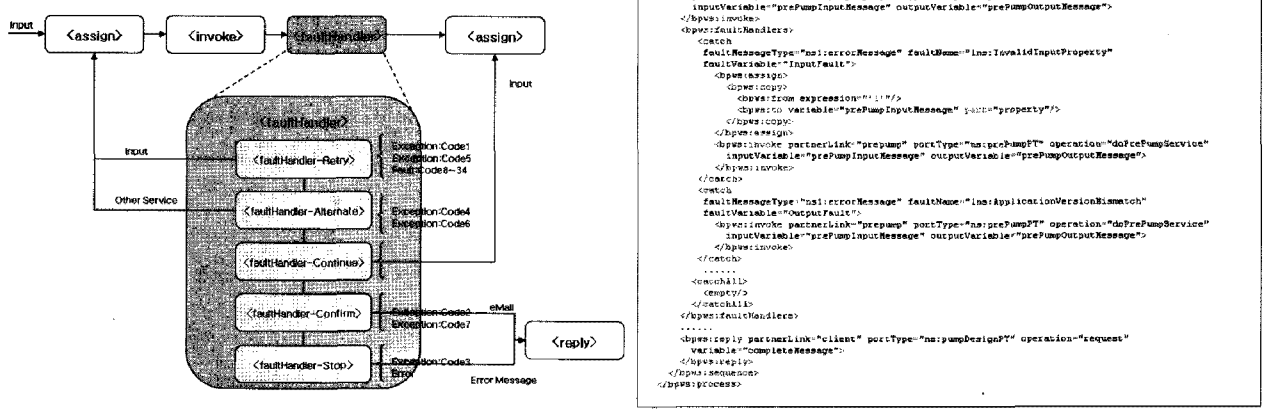


그림 12 Pre_Pump 서비스를 위한 BPEL 프로세스의 확장

6. 결 론

BPEL은 표준 프로세스 모델 언어로, 다양한 서비스를 효율적으로 조합하는 방법을 제시하고 있다. BPEL이 제공하는 다양한 표현 요소를 이용해 서비스 조합에 참여하는 웹 서비스들간의 상호작용을 비즈니스 프로세스로 표현할 수 있기 때문에 BPEL 기반의 비즈니스 프로세스는 최근 증가하고 있는 여러 유형의 시스템 통합에 관한 요구를 수용하기 적당한 방법이다. 여러 서비스의 조합은 단일 서비스에 비해 오류나 예외 상황이 발생할 가능성이 높다. 이러한 점은 서비스 조합을 표현하고 있는 비즈니스 프로세스의 신뢰성에 영향을 주며, 보다 더 강건한 서비스 조합을 구축하기 위해서는 이와 관련된 다양한 테스트의 수행이 필수적이다. 그러나 서비스의 조합을 테스트하는 것은 기존의 단일 애플리케이션을 테스트하는 것보다 훨씬 어렵다. 참여하는 서비스의 수에 따라 다양한 테스트 시나리오가 작성되어야 하며, 경우에 따라서는 상업적인 서비스가 포함될 경우 그 서비스에 대한 추가 비용뿐만 아니라 테스트 목적으로 원활하게 사용하기 어렵기 때문이다. 본 논문에서는 BPEL 기반의 서비스 조합 프로세스의 강건성 테스트를 수행하기 위하여, BPEL 프로세스와 참여하는 웹 서비스를 분석하여 가상의 서비스 수행 환경을 생성하고, 이를 통해 BPEL 프로세스가 얼마나 오류 상황과 예외 상황에 강건하게 대처할 수 있는지를 테스트하는 방법을 제시하였다. 기존의 웹 서비스 조합에서의 테스트는 대부분 기능이나 성능적 측면과 조합 적절성 측면에서 수행되었으며, 강건성이나 신뢰성 측면에 대한 고려는 거의 없었다. 이 논문의 테스트 방법은 기존에 간과되었던 서비스 조합의 신뢰성, 강건성 측면에서의 테스트 방안을 제시하고 있으며, 새로운 BPEL 프로세스를 개발하는 과정, BPEL을 기반으로 다양한 시스템을 통합하는 과정에서 BPEL 프로세스가 다양한 비정상적 상황에 대해 얼마나 강건하고 신뢰할 수 있는지 검증하기 위한 방법으로 활용될 수 있다.

참 고 문 헌

[1] Q.H. Mahmoud, "Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)," http://java.sun.com/developer/technicalArticles/WebServices/soa/, Apr. 2005.

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

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

[4] M.P. Papazoglou, et al. "Service-Oriented Computing," *Comm. ACM*, vol.46, no.10, pp.25-28, 2003.

[5] C. Peltz, "Web services orchestration and choreography," *IEEE Computer*, vol.36, no.8, pp.46-52, 2003.

[6] P. Mayer, et al., "Towards a BPEL unit testing framework," *Proc. of Workshop on Testing, analysis, and verification of web services and applications*, pp.33-42, 2006.

[7] Y. Yuan, et al., "A graph-search based approach to bpel4ws test generation," *Proc. of the International Conference on Software Engineering Advances*, p.14, 2006.

[8] H. Huang, et al. "Automated model checking and testing for composite web services," *Proc. of 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp.300-307, 2005.

[9] A. Bertolino, et al. "The audition framework for testing web services interoperability," *Proc. of 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pp.134-142, 2005.

[10] E. Martin, S. Basu, and T. Xie, "Automated Robustness Testing of Web Services," *Proc. of the 4th International Workshop on SOA and Web Services Best Practices*, 2006.

[11] X. Bai, W. Dong, W.-T. Tsai, Y. Chen, "WSDL-Based Automatic Test Case Generation for Web Services Testing," *Proc. of the 2005 IEEE International Workshop on Service-Oriented System Engineering*, pp.215-220, 2005.

[12] BPEL2.2 Metamodel, http://www.wsper.org/ws-bpel20b.png

[13] WSDL2.0 Metamodel, www.wsper.org/wsdl20.png

[14] S.H. Kuk, I.N. Oh, H.S. Kim, J.-K. Lee, and S.-W. Park, "An e-Engineering Framework Based on Service-Oriented Architecture and Agent Technologies," *Proc. of The 11th International Conference on Computer Supported Cooperative Work in Design*, pp.429-434, 2007.

[15] S.H. Kuk, H.S. Kim, J.-K. Lee, and S.-W Park, "Approaches to Improving Reliability in e-Engineering Framework," *Proc. of 2008 IEEE International Conference on Web Services*, pp.353-360, 2008.

국 승 학
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 1 호 참조

김 현 수
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 1 호 참조