조합된 서비스의 성능 테스트를 위한 에스펙 기반의 WS-BPEL 확장

성동혁[0], 김종필, 홍장의
충북대학교 컴퓨터과학과
{sungdh, kimjp}@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

Aspect-based WS-BPEL Extension for Performance Testing of Composite Service

Dong-Hyuk Sung[0], Jong-Pil Kim, Jane-Eui Hong
Dept of Computer Science, Chungbuk National University

요 약

서비스 지향 아키텍처 기반의 소프트웨어 시스템은 서비스 단위의 컴포넌트 조합에 의해 기능이 수행된다. 이러한 조합된 서비스에 대한 테스팅은 주로 서비스 조합 명세서인 WS-BPEL 명세를 기반으로 기능적인 측면에 초점을 두어 수행 되었다. 최근에 서비스 지향 아키텍처가 임베디드 소프트웨어 분야에 적용되면서 서비스 조합에 대한 성능 관련 요구사항의 중요성이 부각되고 있다. 그러나 기존의 성능 테스트는 각 서비스의 응용 어플리케이션에 성능 측정을 위한 코드삽입을 필요로 함으로써 테스트 비용을 증가시킨다. 또한 시뮬레이션을 통한 성능 테스트가 이루어지기 때문에 정확한 성능을 테스트하기 어렵다. 따라서 본 논문에서는 WS-BPEL을 이용한 조합된 서비스의 효과적인 성능 테스트를 위해 WS-BPEL을 확장하는 방법을 제안한다. 제안하는 확장 기법은 관점 지향 프로그래밍의 Aspect 개념을 WS-BPEL에서 사용할 수 있도록 함으로써 WS-BPEL을 이용한 테스팅에서 조합된 서비스의 성능 테스트를 효과적으로 수행할 수 있는 기회를 제공한다. 또한 본 확장 기법은 성능 테스트뿐만 아니라 다른 비 기능적 요구사항에 대한 테스트에도 활용될 수 있다.

1. 서 론

서비스 지향 아키텍처(Service-Oriented Architecture, SOA)란 컴포넌트 모델의 일종으로 서비스들 간의 계약과 잘 정의된 인터페이스를 통해 서비스라 불리는 어플리케이션들을 상호 연관시키는 기반 환경을 제공한다[1]. 서비스 지향 아키텍처의 가장 큰 특징은 하드웨어, 운영체제, 프로그래밍 언어 등의 구현 환경에 구애 받지 않는 통합적인 환경을 제공하는 것이다. 이러한 특징 때문에 분산 환경에서의 웹 어플리케이션에 많이 적용되고 있다.

서비스 지향 아키텍처는 다양한 비즈니스 요구사항을 수용하기 위해 기존 어플리케이션들을 조합하여 새로운 서비스를 창출할 수 있는 WS-BPEL(Web Service Business Process Execution Language)을 사용한다[2]. WS-BPEL은 웹 서비스의 워크플로우를 표현하기 위해 기존 EA(Enterprise Architecture)에서의 BPEL을 확장한 것이다. 이러한 WS-BPEL은 서비스 간의 호출 관계를 상위 수준에서 정의함으로써, 서비스 간의 조합을 효과적으로 수행할 수 있는 수단이 된다.

서비스 지향 아키텍처에서 서비스 제공은 대부분 WS-BPEL을 통해서 이루어진다. 이것은 WS-BPEL이 잘못 명세 되었을 경우 서비스가 정확하게 제공되지 않을 수 있다는 것을 의미한다. 따라서 WS-BPEL로 기술된 서비스 조합이 서비스 요구사항에 부합하는 지에 대해 테스팅을 수행하는 것이 필요하다. 이러한 테스팅 요구에 따라 WS-BPEL의 서비스 조합을 테스트하기 위한 기존의 연구들이 있었다[3, 4, 5]. 그러나 이러한 연구들은 대부분 서비스 조합의 기능적인 측면에서 단위 테스트나 통합 테스트에 초점을 맞추고 있다.

현재 서비스 지향 아키텍처가 임베디드 소프트웨어 영역에 활용되면서 조합된 서비스의 성능 테스트에 대한 요구가 증대되고 있다[6, 7]. 기존의 서비스 지향 아키텍처 기반 성능 테스트는 각 서비스 내부에 수행 시간을 모니터링 할 수 있는 스크립트 코드를 삽입하여 테스트 프레임워크 상에서 시뮬레이션하는 방식으로 수행하였다[8, 9]. 그러나 이러한 성능 테스트는 모니터링 코드를 테스트 대상이 되는 모든 서비스 응용 어플리케이션에 포함시켜야하고 이를 제어할 수 있는 테스트 프레임워크가 필요하기 때문에 성능 테스트에 대한 많이 비용이 수반된다. 또한 시뮬레이션을 통한 성능 테스트가 이루어지기 때문에 정확한 성능을 테스트하기 어렵다. 따라서 적은 비용으로 조합된 서비스에 대한 성능 테스트를 수행할 수 있는 효과적인 기법이 필요하다. 본 논문에서는 WS-BPEL의 확장을 통해 조합된 서비스의 성능 테스트를 효과적으로 수행할 수 있는 기법을 제안한다. 확장은 WS-BPEL에 관점 지향 프로그래밍의 특성이 적용 가능하도록 이루어졌으며, 이를 통해 WS-BPEL에 횡단 관심사로서 성능 테스팅 기능을 부가적으로 추가할 수 있게 하였다.

* 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2010-(C1090-1031-0001))

본 논문의 구성은 다음과 같다. 2절에서는 제안하는 기법에 관련된 연구들을 살펴보고 3절에서는 본 논문에서 제안하는 WS-BPEL 확장 기법에 대해 기술한다. 4절에서는 제안된 확장 기법이 성능 테스트에 어떤 방식으로 기여하는 지를 살펴보고 5절에서는 예제 시스템을 통한 확장 기법의 적합성을 확인한다. 6절에서는 제안하는 기법의 고려사항에 대해 논의하고 7절에서는 결론과 향후 연구에 대해 기술한다.

2. 관련연구

2.1 WS-BPEL을 이용한 서비스 조합

WS-BPEL은 웹 서비스 기술언어(Web Services Description Language, WSDL)의 명세를 활용하여 프로세스라는 처리 단위로 서비스 조합을 기술한다. 그림 1은 WS-BPEL의 프로세스를 정의하는 구조와 WSDL의 구조를 나타내고 있다[2, 10]. WS-BPEL의 <partnerLink> 절은 참여 서비스를 정의하고 있으며 <sequence>절은 순차적으로 처리할 일련의 행동을 명시하고 있다.

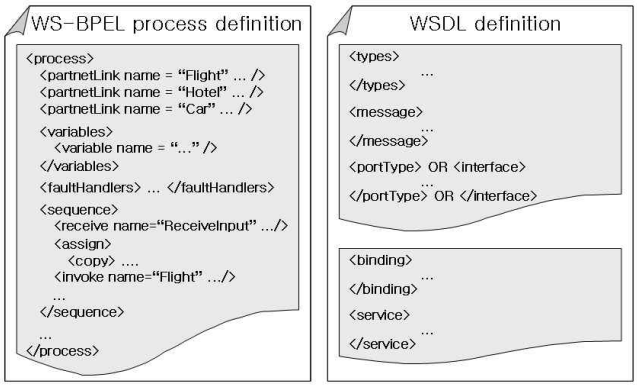

그림 1 WS-BPEL과 WSDL의 구조[1]

이러한 WS-BPEL과 WSDL을 이용한 서비스 조합 과정은 그림 2와 같다. 그림 2는 여행 예약 서비스에 대한 서비스 조합을 보여 준다.

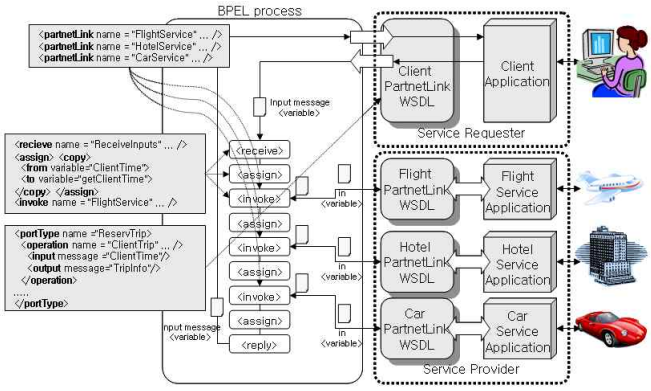

그림 2 여행 예약 서비스에 대한 서비스 조합

이 조합 프로세스에서 참여자는 크게 서비스 요청자와 서비스 제공자로 구성되며 서비스 제공자가 제공하는 여행 예약서비스를 서비스 요청자가 이용한다. 서비스 조합에 각각 참여하는 서비스 제공자는 WSDL과 파트너링크로 정의된다. 이 정의에는 서비스의 위치, 인터페이스의 이름, 입출력데이터, 프로토콜 등에 관한 정보들을 담고 있다. 그림 2와 같이 조합된 예약서비스를 서비스 요청자인 고객이 요청하면 WS-BPEL 실행 엔진을 통해 프로세스가 실행되고 <receive> 요소를 통해 사용자의 요청을 받아들인다. <assign> 요소는 사용자가 제공한 정보를 서비스 제공자의 메시지 타입에 맞게 변경하며 <invoke>에서는 각 제공자의 서비스들을 호출하여 사용자에게 서비스를 제공한다. <reply> 요소는 서비스 처리 결과를 고객에게 전달하는 역할을 수행한다.

위에서 보는 것과 같이 WS-BPEL은 세부적인 서비스들의 실행 흐름을 나타내기 때문에 전체 서비스가 어떠한 순서로 조합되고 실행되는지를 표현한다. 따라서 기존의 많은 연구들이 서비스 지향 아키텍처의 커버리지 테스팅을 수행하기 위해 이러한 WS-BPEL을 활용하고 있다.

2.2 관점 지향 프로그래밍을 이용한 테스팅

관점 지향 프로그래밍은 객체 지향 프로그래밍에서 모듈화 하지 못한 횡단 관심사를 모듈화 하기 위한 방법으로 연구되었다. 이러한 관점 지향 프로그래밍 기법은 Aspect, Pointcut, Advice와 같은 요소를 이용하여 소프트웨어의 핵심 기능에 부가적인 기능을 효율적으로 추가할 수 있는 기능을 제공한다[11]. 이러한 특성으로 인해 분산된 코드들이 자주 발생하는 영역에서 활용되고 있다. 특히 소프트웨어 테스팅에서 사용되는 테스트 코드들을 모듈화하기 위한 기법으로 자주 이용되고 있다. 관점 지향 프로그래밍을 이용한 단위 테스팅 기법[12]에서는 테스트 코드를 Aspect로 모듈화하여 소프트웨어의 각 메소드들을 테스트하였다. Aspect를 이용한 소프트웨어 컴포넌트에 대한 테스팅[13]에서는 테스트 기능을 갖는 Aspect 컴포넌트를 개발하여 각 컴포넌트의 오퍼레이션을 테스트하는 기법을 제안하였다. 이와 같이 관점 지향 프로그래밍은 효율적인 테스트를 위한 수단을 제공한다.

2.3 WS-BPEL을 이용한 조합된 서비스 테스팅

서비스 지향 아키텍처가 다양한 영역에서 활용되면서 많은 테스팅 기법들이 연구되었다. 이러한 테스팅 기법들은 대부분 WSDL이나 WS-BPEL과 같은 산출물을 이용하여 수행되었다.

WSDL을 이용한 테스팅 기법에 관한 연구는 WSDL을 확장하여 메소드의 호출 및 기능들을 절차적으로 명세하고 이를 통한 데이터 플로우 테스팅, 경로 테스팅 등을 수행하였다[4]. WS-BPEL을 이용한 테스팅 기법은 WS-BPEL상에서 조합된 서비스의 경로를 테스팅 하는 프레임워크를 제안한 연구가 있었다[3]. 이 프레임워크는 WS-BPEL을 통해 전체 워크플로우의 경로를 수집하고 각 경로별로 커버리지 테스팅을 수행한다.

이러한 서비스 지향 아키텍처의 테스팅 기법들은 대부분 기능적 측면에서의 서비스 단위 테스트나 서비스 조합에 대한 통합 테스트에 초점을 두고 있으며 성능 기반의 테스팅 연구는 이루어지지 않고 있다. 따라서 본 논문에서는 성능 테스팅을 수행할 수 있는 확장된 WS-BPEL을 제안한다.

2.4 WS-BPEL의 확장

XPDL(XML Process Definition Language)과 WS-BPEL을 통합하는 연구에서는 각 메타 모델의 공통부분을 통합한 하나의 모델을 정의함으로써 확장된 WS-BPEL을 제안하였다[14]. 또한 WS-BPEL에 User와의 상호작용을 표현하기 위해 WS-BPEL 메타모델의 ExtensionActivity 요소를 확장한 연구가 있었다[15].

이러한 확장 연구들은 WS-BPEL로 표현 할 수 없는 서비스 처리 과정을 보완하는 것에 초점을 두고 있으며 관점 지향 프로그램의 적용이나 성능 테스팅에 목적을 둔 확장은 이루어지지 않았다.

3. Aspect 기반의 성능 테스트를 위한 WS-BPEL의 확장

3.1 확장 개요

WS-BPEL과 Aspect를 이용한 응답 시간 관점의 성능 테스트 개념은 그림 3과 같이 기존의 WS-BPEL 명세에 성능 테스트를 위한 Aspect 명세와 타이머 서비스를 추가하는 것이다. Aspect의 PointCut은 WS-BPEL의 오퍼레이션과의 연결점을 나타내고 BeforeAdvice와 AfterAdvice는 각각 수행 시간 측정을 위한 타이머 서비스의 시작과 종료 역할을 수행한다. WS-BPEL 프로세스의 특정 오퍼레이션이 실행될 때 Aspect는 시간 측정을 위한 타이머 서비스의 시작 오퍼레이션을 호출하고 오퍼레이션의 실행이 완료되면 종료 서비스를 호출하여 수행 시간을 측정함으로써 조합된 서비스의 성능을 테스트하게 된다.

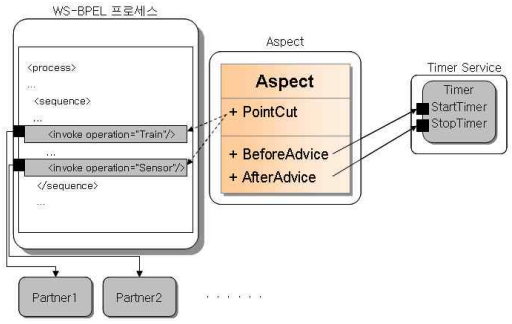

그림 3 Aspect를 이용한 성능 테스트 개념

그림 3과 같은 성능 테스트를 구현하기 위해서는 Aspect를 WS-BPEL 형태로 표현할 수 있는 방법이 필요하다. 즉 관점 지향 프로그래밍에서의 직조 메커니즘이 수행되려면 WS-BPEL 프로세스와 Aspect가 같은 메타 모델 기반으로 기술되어야 하고 Aspect를 통한 부가 기능이 서비스 관점으로 표현되어야 한다. 따라서 관점 지향 프로그래밍의 핵심 요소인 Aspect, Pointcut, Advice를 지원할 수 있도록 WS-BPEL의 확장이 요구된다. 본 논문에서는 이러한 성능 테스트 개념을 지원할 수 있는 WS-BPEL에 대한 확장 방법을 제안하였다.

3.2 Aspect 적용을 위한 WS-BPEL의 확장

이 절에서는 3.1절에서 식별한 Aspect, Pointcut, Advice를 WS-BPEL에서 사용할 수 있도록 WS-BPEL의 메타모델을 확장한다.

3.2.1 Aspect의 확장

WS-BPEL의 ActivityContainer는 일련의 서비스들을 표현할 수 있는 기본 틀이 되는 요소이다. 이러한 측면에서 볼 때 Aspect 또한 성능 테스트라는 부가적 서비스를 포함해야 하기 때문에 ActivityContainer 클래스를 상속받아 정의하였다. WS-BPEL의 Aspect에 대한 확장은 그림 4와 같다. Aspect 클래스는 성능 테스트를 수행할 수 있는 부가 서비스들에 대한 기본 틀 역할을 한다.

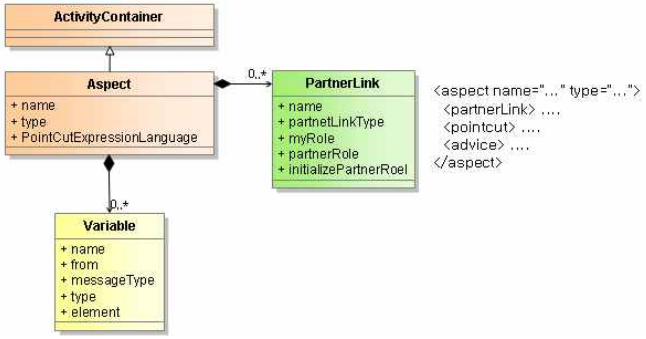

그림 4 Aspect의 확장

Aspect 클래스는 서비스 범위를 표현할 수 있는 PointCutExpressionLanguage와 서비스 타입을 지정할 수 있는 type 속성을 갖는다. 또한 Aspect에는 서비스와 서비스가 사용할 변수에 대한 표현이 필요하기 때문에 PartnerLink 클래스와 Variable 클래스에 대한 포함관계를 갖는다.

3.2.2 Pointcut의 확장

Pointcut은 Aspect 내에서 성능 테스트 대상을 지정하는 역할을 한다. Aspect는 다수의 Pointcut을 가질 수 있고 이를 정의하기 위한 지정자(Designator)를 포함하기 때문에 이들에 대한 확장이 필요하다.

Aspect의 Pointcut과 Pointcut 지정자를 위한 확장은 그림 5와 같다. PointCut의 PointCutExpression은 PointCut 지정자를 표현하는 수단으로써 WS-BPEL의 Expression을 상속받아 확장하였다. 이와 같이 확장된 PointCutExpression을 통해 관점 지향 프로그래밍에서와 같은 유형으로 PointCut 지정자를 표현할 수 있다.

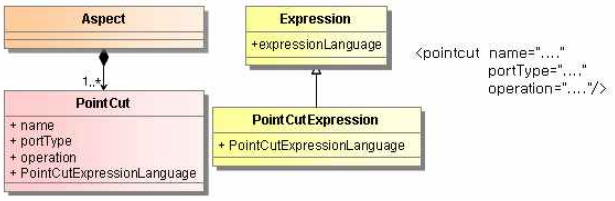

그림 5 PointCut의 확장

3.2.3 Advice의 확장

Advice는 성능 테스트를 위한 타이머 서비스를 호출하는 행위를 수행한다. WS-BPEL에서는 ExtensionActivity라는 요소를 통해 WS-BPEL에 부가적인 행위를 추가할 수 있는 기능을 제공한다. 이러한 ExtensionActivity 요소를 상속받아 성능 측정을 수행하는 Advice를 그림 6과 같이 확장하였다.

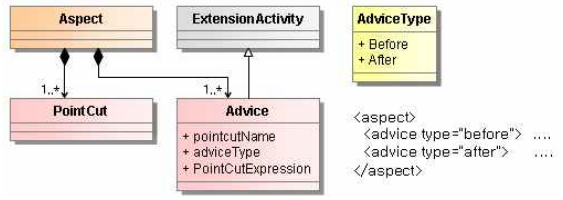

그림 6 Advice의 확장

Aspect는 하나 이상의 Advice를 갖기 때문에 그림 6과 같은 포함관계를 갖는다. 또한 각각의 Advice는 선언된 PointCutName을 기점으로 AdviceType 클래스의 Before와 After 속성에 따라 실행한다. 이는 타이머 서비스의 이전/이후의 실행기점을 각각 표현한다. Aspect 요소들을 반영한 확장된 WS-BPEL 메타모델은 생략하였다.

4. 성능 테스팅을 위한 확장된 WS-BPEL의 적용

4.1 성능 테스트를 위한 Aspect의 작성

확장된 WS-BPEL을 이용하여 성능테스트를 수행하기 위해서는 타이머 서비스를 제공하는 Aspect가 작성되어야한다. 그림 7은 이러한 Aspect의 작성과정을 나타낸다. Aspect의 작성을 위해 먼저 WS-BPEL의 연결점(Join Point)의 식별이 선행되어야 한다. 연결점의 식별은 WSDL의 서비스 세부 명세와 WS-BPEL에서 호출, 수신, 답장 같은 WS-BPEL의 Basic Activity 부분이 해당된다. 연결점에 해당하는 오퍼레이션들은 이후 성능 테스트의 대상이 된다.

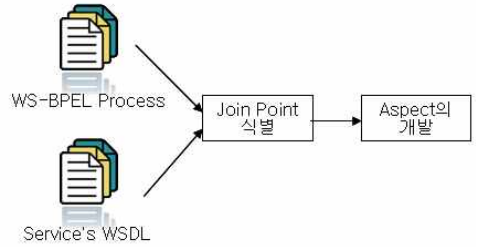

그림 7 WS-BPEL을 이용한 Aspect 작성

식별된 연결점들은 <pointcut>을 통해 정의한다. <pointcut>에는 정형적인 표현 방법으로 테스트 대상 오퍼레이션을 정의한다. <advice>에는 "before"와 "after" 타입을 사용하여 성능 테스트를 위해 타이머 서비스를 호출하는 기능을 기술한다. 이렇게 정의된 Aspect는 그림 8과 같다.

```
<process ...> ...
  <aspect name="Aspect" type="composition" ... />
    <partnerLink name="TimerController" />
    <pointcut name="SamplePointcut"
              portType="SampleInterface"
              operation="SampleOperation" />
    <advice type ="before" pointcutName="SamplePointcut">
      <invoke partnerLink="TimerController"/
              operation="StartTimer" ... /> </advice>
    <advice type ="after" pointcutName="SamplePointcut">
      <invoke partnerLink="TimerController"/
              operation="StopTimer" ... /> </advice>
  </aspect> ...
</process>
```

그림 8 성능 테스트를 위한 Aspect

그림 8에서 <aspect>의 type 속성은 성능 테스트를 수행할 대상 서비스가 조합된 서비스 또는 단일 서비스인지를 표현하는 것으로 composition, single과 같은 값을 갖는다. 연결점을 나타내는 <pointcut>의 portType 속성은 테스트 대상에 접근하기 위한 포트를 명시하고 operation 속성은 대상 서비스의 포트를 통해 수행되는 서비스 오퍼레이션을 나타낸다. <advice>의 <invoke>에 명시된 partnerLink 속성에는 성능 측정을 위한 타이머 서비스가 정의되고 operation 속성에는 타이머 서비스의 실제적인 기능을 수행하는 타이머 시작, 타이머 종료 오퍼레이션이 수행된다.

4.2 Aspect를 이용한 WS-BPEL 기반의 성능 테스팅

확장된 WS-BPEL을 통한 성능 테스팅은 그림 9와 같은 과정으로 이루어진다. 먼저 WS-BPEL 프로세스와 성능 테스트를 위한 Aspect의 직조가 수행된다. 직조를 통해 생성된 성능 테스팅이 가능한 WS-BPEL은 서비스 지향 아키텍처 프레임워크 내에서 테스트 드라이버를 통해 성능 테스트를 수행하게 된다.

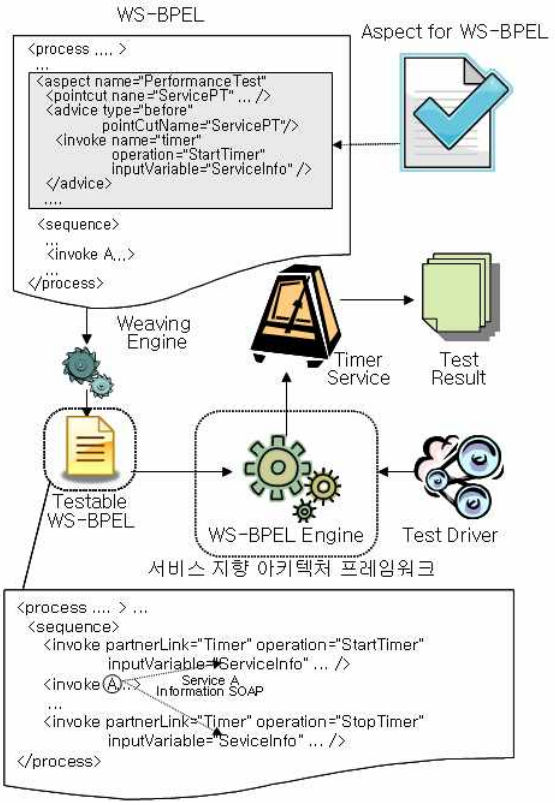

그림 9 Aspect를 이용한 성능 테스팅 수행 과정

테스트 결과는 타이머 서비스를 구현한 어플리케이션에서 수집되어 저장된다.

5. 예제 시스템의 적용

제안한 확장 기법이 성능 테스트에 활용될 수 있는 지를 예제 시스템을 통해 확인하였다. 예제 시스템은 그림 10과 같이 서비스 지향 아키텍처로 개발된 열차 관리 시스템[7]으로, 열차 운용에 필요한 다양한 하드웨어 디바이스들과

어플리케이션으로 이루어져있다. 대부분의 어플리케이션들이 하드웨어 디바이스에 내장되어 실시간 처리가 필요한 서비스를 제공하고 있기 때문에 성능 요구사항에 대한 만족여부가 중요하다. 또한 이 시스템은 서비스 지향 아키텍처를 통해 열차의 교체와 어플리케이션의 추가 및 변경에 대해 유동적으로 대응할 수 있다[16]. 그림 10은 예제 시스템에서 구축하고 있는 서비스 지향 아키텍처의 모습을 보여준다.

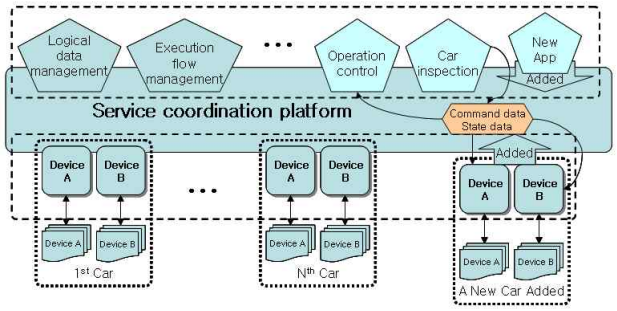

그림 10 열차 관리 시스템 [7]

예제 시스템에 대한 WS-BPEL 프로세스는 그림 11과 같다. 이것은 하나의 열차가 추가되면서 일어나는 일련의 프로세스를 표현하고 있다.

```
<process name="TrainCarManagementSystem">
  <partnerLink name="LocalDataManager" ... />
  <partnetLink name="CarInspection .../>
  <partnetLink name="OperationControl" ... />
  ...
  <partnerLink name="NewTrainCar" ... >          ..... ①
  <variables>
    <variable name="CarDevice"                   ..... ②
          messageType="car:devices" ..."/>
  </variables>
    ...
  <sequence>
    ...
    <receive name="ReceiveCarDevice" ... />  ..... ③
    <assign> <copy> ... </copy>     </assign>
    <invoke name="InspectionNewCar"              ..... ④
      partnetLink="CarInspection"
      portType="Ins:InspectionInterface"
      operation="CarInspection"
      inputVariable="getOperationRequestMessage"
      outputVariable="getOperationResponsMessage"/>
    <invoke name="RequestOperation"              ..... ⑤
      partnetLink="OperationControl"
      portType="ops:OperationListInterface"
      operation="GetOperationList"
      inputVariable="getOperationRequestMessage"
      outputVariable="getOperationResponsMessage"/>
      ....
  <sequence>
</process>
```

그림 11 예제 시스템의 WS-BPEL 프로세스

①의 <partnerLink>에서는 추가 된 열차가 정의된다. ②는 열차의 추가에 의한 장치 정보를 위한 변수가 <variable>에 정의되고 이러한 정보는 ③의 <receive>, <assign>을 통하여 CarInspection 서비스에 전송된다. ④의 CarInspection 서비스는 추가된 열차가 전체 시스템에 적합한 지를 확인하는 기능을 수행하고 ⑤의 OperationControl 서비스는 추가된 열차를 모니터링 하여 제어하는 기능을 수행한다.

예제 시스템의 WS-BPEL 프로세스에 대한 성능 테스트 기능을 제공하는 Aspect는 그림 12와 같다.

```
<process name="TrainCarManagementSystem">
  ...
  <aspect name="TestAspect">
    <partnerLink name="Timer" partnerLinkType .... />
    <variables>
      <variable name="ServiceInfo"
                messageType="var:ServiceInfo" .... />
    </variable>
    <pointcut  name="OperationTimePC"        ..... ⑥
               type="composition"
               portType="OperationListInterface"
               operation="GetOperationList" />
    <advice type ="before"                          ..... ⑦
            pointcutName="OperationTimePC">
      <assign> <copy> <from> pointcut.info </from>
        <to> ServiceInfo </to> </copy> </assign>
      <invoke partnerLink="Timer"
        operation="StartTimer"
        inputVariable="ServiceInfo"
        portType="msg:TimerInterface"/> </advice>
    <advice type ="after"                           ..... ⑧
            pointcutName="OperationTimePC">
      <invoke partnerLink="Timer"
        operation="StopTimer"
        inputVariable="ServiceInfo"
        portType="msg:TimerInterface"/>    </advice>
  </aspect>
  ...
```

그림 12 예제 시스템의 <Aspect>

확장된 WS-BPEL의 특징을 이용하여 WS-BPEL 프로세스 내부에서 <Aspect> 라는 요소를 추가로 사용하여 기술되었다. ⑥의 <pointcut>에 기술된 테스트 대상은 OperationControl 서비스이고 이에 대한 성능 측정을 위한 타이머 서비스는 ⑦, ⑧의 <advice>를 통해 호출된다.

예제 시스템의 WS-BPEL 프로세스는 Aspect와의 직조 과정을 거쳐 성능 테스트를 수행하게 된다. 직조 과정이 이루어진 성능 테스트가 가능한 WS-BPEL 프로세스는 그림 13과 같다.

```
  ...
  <partnerLink name="Timer" ... /> ...          ..... ⑨
  <variable>
    <variable name="ServiceInfo" ... /> ...
  <sequence>
    ...
    <assign>
      <copy> <from>
        <literal> OperationControl </literal> </from>
      <to> ServiceInfo.Service </to>   </copy>
      <copy> <from>
          <literal>GetOperationList</literal> </from>
      <to> ServiceInfo.operation </to> </copy>
    </assign>
    <invoke partnerLink="Timer"
      operation="StartTimer"
      inputVariable="ServiceInfo"
      portType="msg:TimerInterface"/>
    <invoke name="ControlTrain"                 ..... ⑩
      partnetLink="OperationCtr"
      portType="ops:OperationList"
      operation="GetOperationList"
      inputVariable="getOperationRequestMessage"
      outputVariable="getOperationResponsMessage"/>
    <invoke partnerLink="Timer"
      operation="StopTimer"
      inputVariable="ServiceInfo"
      portType="msg:TimerInterface"/>
      ...
```

그림 13 직조된 Testable WS-BPEL

직조된 WS-BPEL은 테스트 대상인 ⑩의 OperationControl 서비스가 수행될 때마다 ⑨의 타이머 서비스를 호출하여 수행 시간을 측정하게 된다. 각 서비스의 테스트 결과는 타이머 서비스 어플리케이션에서의 수집 과정을 거쳐 테스터에게 제공된다.

6. 제안하는 성능 테스팅 기법에 대한 고려 이슈

확장된 WS-BPEL을 이용하여 성능 테스트를 수행할 때 고려해야 하는 이슈들은 다음과 같다.

- 성능 테스트를 수행할 때 타이머 서비스에 의한 수행 시간 지연은 어떻게 처리되는가? 분산된 서비스들이 WS-BPEL 엔진에 의해 조합되어 수행될 때 타이머 서비스 수행에 의한 시간 지연이 발생할 수 있다. 즉 타이머 서비스 수행에 따른 시간 지연으로 인해 정확하지 못한 성능 측정값을 얻을 수 있다는 것이다. 이러한 시간 지연은 타이머 서비스를 분산된 서비스가 아닌 WS-BPEL 엔진 내부에 포함시키고 호출 메시지에 의한 비동기적인 수행으로 해결될 수 있다.

- 성능 테스트가 네트워크 상황을 고려하여 수행되는가? 성능 테스트는 구성 서비스 간의 접근이 가능한 네트워크 환경이 구성된 이후에 수행된다. 따라서 수행 시간 측정 결과는 네트워크상에서 송신되는 서비스 호출 메시지와 수신되는 서비스 실행 완료 응답 메시지를 기반으로 이루어지기 때문에 네트워크 상황이 반영될 수 있다.

- 성능 테스트를 위해 요구되는 테스트 데이터들은 어떻게 도출되는가? 조합된 서비스에 대해 성능 테스트를 수행하기 위해서는 서비스 조합에 참여하는 서비스들의 실행을 일련의 순서로 진행시킬 수 있는 테스트 시나리오가 필요하다. 이러한 테스트 시나리오는 서비스 입력 값들의 조합으로 이루어지며 테스트 데이터 도출에 활용될 수 있다.

7. 결론

임베디드 소프트웨어 분야에 서비스 지향 아키텍처가 적용되면서 조합된 서비스에 대한 성능 관련 요구가 증대되고 있다. 그러나 기존 조합된 서비스에 대한 테스팅에 관련된 연구들은 대부분 WS-BPEL을 활용한 기능적인 측면에서의 단위 테스트나 통합 테스트만을 수행하고 있다. 따라서 본 논문에서는 성능 테스트를 수행할 수 있는 확장된 WS-BPEL을 제안하였다. 제안된 확장은 WS-BPEL에 관점 지향 프로그래밍의 특성이 적용 가능하도록 이루어졌으며, 이를 통해 WS-BPEL에 횡단 관심사로서의 성능 테스트 기능을 부가적으로 추가할 수 있도록 하였다. 예제 시스템을 통해 확장된 WS-BPEL이 성능 테스트를 가능하도록 활용될 수 있음을 확인하였다. 제안된 확장은 관점 지향 프로그래밍과 WS-BPEL의 특성을 활용하여 적은 노력으로 분산되어 있는 조합된 서비스들의 성능을 효과적으로 테스트할 수 있다는 장점을 제공한다. 향후에는 제안된 테스팅 기법의 실용성 확인을 거쳐 자동화된 성능 테스트 도구를 개발할 계획이다.

참고 문헌

[1] Thomas Erl, "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall PTR Upper Saddle River, 2005.

[2] OASIS, WS-BPEL 2.0(Web Services-Business Process Execution Language), http:\\docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[3] T. Lertphumpanya, T. Senivongse, "A basis path testing framework for WS-BPEL composite services", Proceedings of the 7th WSEAS, 2008.

[4] W. T. Tsai, et. al., "Extending WSDL to Facilitate Web Sof 7th IEEE International Symposium on High Assurance System Engineering, pp.171-172, 2002.

[5] C. Bartolini, et. al., "Whitening SOA testing", Proceeding of ACM SIGSOFT symposium on The foundations of software engineering, pp.161-170, 2009.

[6] E. Zeeb, et. al., "WS4D: SOA-Toolkits making embedded systems ready for Web Services", Proceedings of the Open Source Software and Product Lines Workshop, 2007.

[7] Kakumoto, et. al., "Component Oriented Software Framework for Train Car Systems", Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, 2005.

[8] T. T. Cheng and C. H. Fu, "On the Development of Software Tools for Testing Web Services", Proceedings of International Conference on Internet Computing, 2004.

[9] S. Chandrasezkaran, et. al., "Performance Analysis and Simulation of Composite Web Services", Journal of Electronic Markets, Vol.13, Issue 2, pp.120-132, 2009.

[10] W3C, WSDL 2.0 (Web Services Description Language Version 2.0), http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626, 2007.

[11] G. Kiczales, et. al., "Aspect-Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming, 1997.

[12] X. Guoqing and Z. Yang, "A Novel Approach to Unit Testing : The Aspect-Oriented Way", Proceedings of International Symposium on Future Software Technology, 2004.

[13] J. Bruel, et. al., "Using Aspects to Develop Built-in Tests for Components", Proceeding of 6th International Conference on the Unified Modeling Language, 2003.

[14] T. Hornung, et. al., "Integration of heterogeneous BPM Schemas: The Case of XPDL and BPEL", Proceeding of The 18th Conference on Advanced Information System Engineering, 2006.

[15] Jonathan Lee, et. al., "BPEL extensions to user-interactive service delivery" Journal of Information Science and Engineering, Vol.25, pp.1427-1445, 2009.

[16] N. Komoda, "Service Oriented Architecture (SOA) in Industrial Systems", Proceeding of IEEE International Conference on Industrial Informatics, pp.1-5, 2006.