SOA 의 변화하는 서비스를 위한 웹서비스 활용 방안

김형우
삼성 SDS IT 컨설팅실 웹서비스/SOA 사업단
e-mail : dolsokim@samsung.com , dolso25@naver.com

The Analysis of Web Services Tech. and New Suggestion for SOA with Changing Services

HyoungWoo Kim
SamsungSDS IT Consulting Div.
Web Servcies/SOA Business Team

요 약

웹서비스는 수년 전부터 SOA 를 구현하기에 적합한 기술로 널리 인정되어 많은 SOA 관련 프로젝트들에서 사용되고 있다. 본 논문에서는 SOA 를 구현한 표준 웹서비스가 SOA 의 운영과정에서 서비스들에 변화가 가해짐으로 인해 발생될 수 있는 문제들을 살펴본 후, 그 문제들을 고려하여 변화하는 서비스들에 잘 적응하게 하기 위한 웹서비스의 소비. 구축, 설계의 방안을 제시한다.

1. 서론

1.1 웹서비스와 SOA

웹서비스는 웹 전송 프로토콜을 통해 XML 기반의 메시지를 주고 받으면서 특정 작업을 수행하는 것이라고 볼 수 있으며, 그 구성 기술들의 범용성 덕분에 플랫폼 독립성을 보장받아 다양한 플랫폼 사이에서 문제없이 사용 될 수 있었다. 그리고, SOA 는 한 조직의 시스템 내부 구조를 '서비스'라는 기능 묶음의 단위와 그들간의 관계를 정리하여 구조화하는 개념이다. SOA 는 이론상으로는 여러 가지 기술로 구현될 수 있으나 현재로서는 사실상 웹서비스만이 이상적인 SOA 를 구현할 수 있는 수단으로 인식되고 있다.

웹서비스가 SOA 에 적합하다고 판단되는 가장 큰 이유로는 웹서비스가 어느 플랫폼에서나 활용 가능한 범용의 기술을 활용하고 있다는 것과, 서비스의 제공자와 서비스 소비자의 사이에는 비즈니스적인 상관관계 이외에는 어떠한 기술적인 의존이나 전제조건이 필요하지 않는 느슨한 연결(loosely coupled)의 속성을 가지는 것을 들 수 있다

2. 본론

2.1 SOA 운영의 복병 : 변화하는 서비스

SOA 환경에서 웹서비스를 운영하는 과정에서, 서비스 본연의 비즈니스적인 의미는 변하지 않으면서 서비스 자체가 변경되는 경우가 종종 있다. 서비스의 설계자들이 모든 요구사항과 환경, 경영 로드맵 등을 충분히 고려하여 서비스들과 오퍼레이션, 그리고 그 메시지들을 신중하게 설계하며, 그 설계에 따라 서비스들이 제작되지만, 그래도 장기간 운영되는 동안에는 내외부의 돌발변수(정부의 정책 변화, 신규 비즈니스 도입 등)가 생기게 마련이기 때문이다.

이 경우, 서비스에 변화가 생긴다면, 그 변화의 범위로는 서비스들의 모든 속성에서부터 내부 로직까지도 모두 포함할 수 있다. 예를 들어 한 서비스의 서비스명이 바뀔 수 있으며, 서비스에 오퍼레이션이 추가되거나 삭제, 변경 될 수도 있고, 한 오퍼레이션의 input/output 메시지의 XML 스키마가 변경 될 수도 있다.

결국, 서비스자체의 변화는 피할 수 없는 것이고, 그 변화는 전방위로 일어날 수 있는 것이라는 것이며, 이 때문에 서비스 제공자는 항상 변화를 사전에 염두에 두고 서비스에 대한 인프라를 마련해야 한다.

2.2 서비스의 변화에 취약한 웹서비스

서비스 제공자의 입장에서, 서비스의 변화에 대한 요구가 제기되어 서비스 자체에 대한 변화를 수행하는 작업 자체도 큰 문제라고 볼 수 있지만, 더 큰 문제는 어떻게 서비스를 사용하고 있는 사용자들이 대처하도록 할 것이냐에 있다. 일반적인 웹서비스 소비 프로그램들은 서비스 자체의 변화에 취약하며, 서비스의 변화는 곧 서비스 사용불가를 의미하는 것이 되기 때문인데(2.3 장 참고) 더구나 서비스 사용 시스템들의 목록 및 담당자 연락처를 관리하지 않는 공개 서비스 환경의 경우는 서비스의 변화사항을 서비스 사용자 측에 알려줄 수 있는 방법이 없기 때문에 더욱 심각한 문제가 될 것이다. 왜냐하면, 서비스의 변화를 통보 받지 못한 사용자는 다음 번의 서비스 호출 시까지 사전에 변화를 파악하고 대처할 수 있는 방법이 없고, 결국 서비스 호출실패를 경험하게 될 것이기 때문이다.

이는 한마디로 말해서 서비스의 변화관리가 어렵다는 것이며, 웹서비스 환경도 오래 전에 주류를 이루었던 Client/Server 환경에서의 소프트웨어 배포문제와 유사한 문제점을 가지고 있는 것처럼 보인다. 하지만, 웹서비스에서는 XML 을 기반으로 하고 있기에 이런 문제를 벗어날 수 있는 해결책을 이미 가지고 있으며,

단지 몇 가지만 주의하여 서비스를 제작하면 이 문제를 피할 수 있다.

2.3 서비스 변화시의 문제 요인 : 자동화된 바인딩

웹서비스의 서비스들은 특정한 플랫폼 및 특정 프로그래밍 언어로 작성된 내부 시스템의 한 모듈에서 사용하는 오브젝트들을 각각 하나의 XML 문서로 변환하여 웹서비스 XML 문서 속에 포함시켜 상대방에게 전달하며, 외부에서 전달된 XML 문서도 내부 시스템에 적합한 객체로 변환하여 그 시스템의 내부 모듈이 사용할 수 있게 하는데, 이런 내부 객체와 XML 문서 간 양방향의 변환 과정을 바인딩이라고 한다.

웹서비스의 붐이 일던 2000 년 초 당시, 웹서비스를 사용하거나 제작하는 과정에서 가장 손이 많이 가고 문제가 많이 생기는 부분이 바로 이 바인딩 과정이었으며, 많은 웹서비스 관련 오픈소스 단체들과 밴더들은 이 바인딩을 자동으로 처리해주기 위해서 여러 가지 스팩들과 표준, 라이브러리들을 개발해 왔다. 이 바인딩 기술들의 탄생 덕에 많은 IT 종사자들이 쉽게 웹서비스 기술에 접근할 수 있게 되었고, 웹서비스의 제작과 사용에서 많은 작업을 덜게 되어 생산성이 향상 되었던 것은 사실이지만, 이 기술로 인해 잃은 것도 있다. 바로 송수신 XML 메시지의 구조에 대한 어떠한 변화도 용납하지 못하게 된 것이다.

그 이유는 XML 문서를 내부 객체로 변환(Unmarshal)하는 바인딩이 수행되기 위해서는 변환 대상이 되는 XML 문서가 유효(valid)해야 한다는 전제조건 때문이다. 이는 곧 정형(well-formed)한 XML 메시지라 하더라도 유효한 XML 메시지가 아니라면 처리할 수 없게 된다는 것으로, 다시 말해, 서비스를 처음 사용할 당시에 조회한 WSDL 문서에 정의된 XML 스키마에서 조금이라도 벗어나는 웹서비스 문서는 내부 객체로 바인딩 되지 못하고 에러를 발생하게 된다는 의미이다. 그래서, 2.2 장에서 논한 바와 같이 서비스들의(즉, WSDL 문서의) 변화가 발생하면, 자동화된 바인딩을 사용하는 기존의 웹서비스 사용자 모듈들은 모두 사용할 수 없게 되는 것이다. 결국, 자동화된 바인딩이 웹서비스의 변화에 대한 적응성을 떨어뜨리는 요인이라는 것이다.

예를 들어, SOA 를 구성하고 있는 한 서비스의 응답메시지에 포함된 element 의 XSD 정의가 <표 1>과 같이 정의되었다면,

<표 1> 개인정보 XSD 예

```
<complexType> <all>
<element name="age" type="xsd:string" />
<element name="addr" type="xsd:string" />
</all> </complexType>
```

그 응답 메시지는 <표 2>와 같이 구성될 것이다.

<표 2> 개인정보 element 의 예

```
<age>12</age>
<addr>서울시</addr>
```

이 서비스가 시작된 이 후, SOA 환경이 점점 거대해지고 운영이 지속됨에 따라 이 서비스가 점점 많은 사용자들에 의해 호출되게 되었을 때. 만약, 새로 이 서비스를 사용하려고 하는 사용자측에서 이 서비스의 응답 메시지에 <표 3>과 같이 tel(전화번호) element 를 추가하고 서비스를 수정하였다면,

```
<complexType>
<all>
<element name="age" type="xsd:string" />
<element name="tel" type="xsd:string" />
<element name="addr" type="xsd:string" />
</all>
</complexType>
```

<표 3> 변경된 개인정보 XSD 예

그때부터는 <표 4> 와 같은 메시지가 응답될 것이다.

```
<age>12</age>
<tel>02-333-3333</tel>
<addr>서울시</addr>
```

<표 4> 변경된 개인정보 element 예

이때, 변경된 element 에는 단순히 부가적인 자식 element 가 하나 생겼을 뿐이고, 변경된 element 도 여전히 하나의 정형한 XML 문서이다. 그리고 기존의 자식 element 들을 모두 포함하고 있기 때문에 기존과 똑같이 활용할 수 있는 문서이다. 하지만, 기존의 [표 1]의 정의를 보고 자동화된 바인딩을 사용하는 웹서비스 호출 모듈을 생성한 웹서비스 소비자에서는 이 변화를 감당하지 못하고 [표 4]의 문서에 대해 유효하지 못한 문서임을 알리는 에러를 발생시키게 될 것이다.

2.4 자동화된 바인딩을 사용하지 않는 예

2.5 에서 보인 예의 상황에서 기존에 사용되던 웹서비스 호출 모듈들이 자동화된 바인딩을 사용하지 않는다면 어떻게 변화된 후의 상황에서도 응답 메시지를 제대로 처리 할 수 있게 되는 것일까?

만약 기존에 사용되던 웹서비스 호출 모듈이 [표 2]의 element 에 대한 처리방법을 다음의 <표 5>와 같이 하였다면,

<표 5> 자동화된 바인딩을 사용하지 않는 방식

1. 응답 XML 문서를 DOM 객체로 읽는다.
2. DOM 파싱이나 XPATH, XQUERY 등으로 age element 아래의 text 를 읽어서 'Int Age'의 내부 변수에 할당한다.
3. DOM 파싱이나 XPATH, XQUERY 등으로 addr element 아래의 text 를 읽어서 'String Addr'의 내부 변수에 할당한다.

이 [표 5] 처리방식은 [표 4]의 변경된 XML 문서에서도 그대로 활용될 수 있다. 새로 추가된 tel element 는 기존의 웹서비스 호출 모듈들에서는 사용되지 않기 때문에 있든 없든 영향을 미치지 않기 때문이다. 즉, 수동으로 바인딩을 수행하는 경우에는 2.5 의 예에서 보인 서비스의 변화에도 어떠한 영향도 받지 않는 것이다.

2.5 변화하는 SOA 환경을 위한 웹서비스 활용 가이드

이렇듯, 자동화된 바인딩의 사용은 서비스 소비자들이 서비스의 변화에 적응하는 것을 어렵게 한다. 본 논문에서는 이 문제에 대한 해결책으로 서비스 소비자들이 자동화된 바인딩을 사용하지 않게 하거나 서비스 제공자가 자동화된 바인딩을 고려하여 서비스의 변경시에도 변경이전의 서비스들을 유지시켜 줄 수 있도록 서비스의 버전을 관리하는 방안을 제시하며, 이런 경우 서비스 설계시의 고려사항도 제시한다.

(1) 가이드 1 : 서비스 소비 가이드

SAAJ, JAX-RPC, JAX-WS 의 표준 API 스펙을 따르는 웹서비스 엔진을 사용하여 특정 웹서비스를 호출하는 모듈을 만들 때 그 방식은 크게 SAAJ , JAX-RPC DII(Dynamic Invocation Interface), JAX-RPC SEI(Service Endpoint Interface), JAX-WS 의 4 가지가 있다.

이 4 가지 방식에 대한 속성을 정리하면 다음 [표 6]과 같다.

<표 6> 서비스 소비 방식별 특징

	WSDL 사용 초기화	자동화된 바인딩	자동 코드 생성	서비스 변화 적응성
SAAJ	X	X	X	O
DII	X	△	X	△
SEI	O	O	O	X
JAX-WS	O	O	O	X

이 표에서 보듯이 자동화된 바인딩을 사용하지 않는 방식은 SAAJ 방식과, JAX-RPC DII 방식이며, 나머지 두 방식은 모두 원격지의 WSDL 문서로 자동화된 바인딩을 수행하는 코드를 자동생성하며, 런타임 시에도 원격지의 WSDL 문서를 참조하는 서비스 변화에 대한 적응성이 떨어지는 방식이다.

SAAJ 방식은 웹서비스를 호출하는 SOAP 메시지와 응답으로 받은 SOAP 메시지를 DOM 과 같은 일반적인 XML 문서를 다루는 방식으로 메시지를 구성하거나 처리 할 수 있게 해 주며, JAX-RPC DII 방식은 프로그래밍 객체와 XML 문서의 바인딩을 처리해 주는 Serializer 및 Desterilizer 를 개발자가 직접 작성할 수 있도록 해 준다. 이 두 방식을 따르게 되면, 런타임시 WSDL 문서를 참조하거나, 자동화된 바인딩을 수행하지 않는 장점이 있기 때문에 서비스의 변화에 따라 일일이 웹서비스 호출 모듈을 변경하지 않으려면 SAAJ 방식이나 JAX-RPC DII 방식으로 웹서비스를 호출해야 할 것이다.

그러나, 반면에 이 두 방식을 사용할 경우 WS-*로 표현되는 보안이나 전송 보장 등의 부가 웹서비스 표준을 적용하기가 복잡해지는 단점이 있기에 꼭 WS-* 표준을 적용해야 하는 필요가 있는 경우에는 3,4 의 방식을 사용하되, WSDL 문서를 로컬 파일시스템에서 참조하게 하고 응답 메시지에 대해 XSLT 변환을 수행하는 client response handler 를 추가로 작성하는 것으로 자동화된 바인딩의 문제를 피해갈 수도 있다.

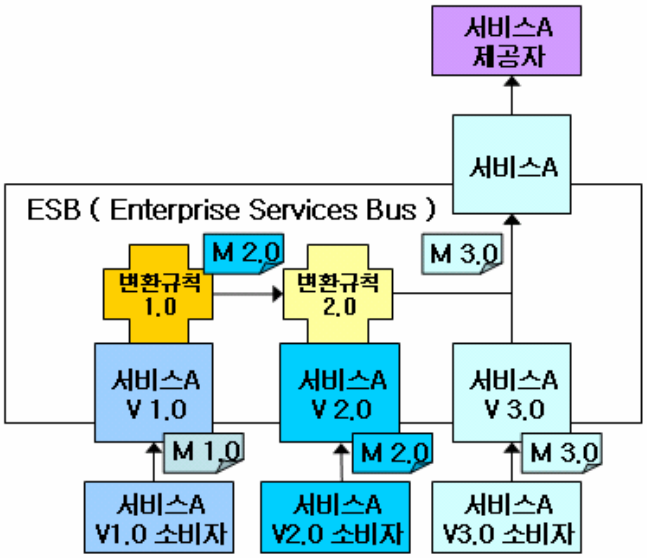
(2) 가이드 2 : 서비스 구축 가이드

서비스의 구축 및 수정 시에도 서비스 변화에 따른 문제를 고려한 해결 방법이 있는데, JAX-RPC/JAX-WS 표준의 핸들러를 사용하거나 WSM(Web Services Management) 또는 JBI(Java Business Integration, JSR208) 표준을 따르는 ESB(Enterprise Services Bus)를 통해 메시지 전처리 단계를 두어 요청 메시지가 들어오기 직전이나 응답메시지가 나간 직후에 메시지에 특정 규칙으로 변화를 가할 수 있는 환경을 구축하는 것이 그것이다.

JAX-RPC 또는 JAX-WS 를 따르는 웹서비스 엔진을 사용하여 웹서비스 제공자를 구축하면, 요청 웹서비스 메시지가 자동화된 바인딩을 거쳐 프로그래밍 객체로 변환되기 전에 '핸들러'라는 일종의 전처리기를 배치할 수 있으며 그 내부에서는 SAAJ 및 XSLT 등으로 메시지의 구조 및 내용을 조정할 수 있다.

그리고 같은 목적으로 웹서비스 제공자와 소비자사이에 배치되는 별도의 솔루션인 ESB 를 사용하는 방법이 있는데, 여기서 ESB 를 사용하면 핸들러 방식보다 더 체계적인 변화내용 및 규칙의 관리가 가능하게 되어 지속적인 서비스의 변화에도 전처리 과정 자체의 혼란을 덜어줄 수 있다.

(그림 1)은 ESB 를 사용하여 서비스 자체의 변화를 관리하고 있는 모습을 예시로 보여주고 있는데, 여기서 예시로 보이는 서비스 A 는 두 차례의 변화를 겪었으며, 처음 변화를 겪었을 때 버전 2.0 의 서비스가 추가되고, 두 번째 변화가 생겼을 때, 버전 3.0 의 서비스가 차례로 생겨났다. 여기서, 서비스의 최초 생성 당시의 소비자, 즉 '서비스 A 버전 1.0 소비자'는 여전히 '서비스 A 버전 1.0'을 호출 할 수 있으며, 그 버전 1.0 의 요청메시지(M1.0)는 '변환규칙 1.0' 과 '변환규칙 2.0' 차례로 적용 받음으로써 버전 3.0 의 메시지(M3.0)로 변환 되어 변화된 서비스 A 를 이전과 같은 방식으로 호출 할 수 있음을 보여준다.



(그림 1) ESB 를 사용한 웹서비스 변화관리

(3) 가이드 3 : 서비스 설계 방안

그러나 가이드 1 이나 2 를 따른다고 해서 모든 서비

스의 변화에 적응할 수 있는 것은 아니다. 서비스의 변화가 단지 메시지 구조나 오퍼레이션의 추가에 그치지 않고 서비스의 비즈니스적인 의미가 변경되는 수준까지 심각하다면, 해당 서비스는 폐지되고 새로운 서비스가 제공되어야 할 것이며, 서비스의 메시지 구조만 변경된다 하더라도, 이전 버전의 서비스에서 필수로 존재하고 사용되던 element 가 삭제되는 변화에서는 서비스의 버전증가가 아닌 별도의 서비스를 구성하는 것이 옳은 선택일 수 있기 때문이다.
즉, 서비스 변화의 설계는 <표 7>의 원칙을 따라야만 한다.

<표 7> 서비스 변화 설계 원칙

1. 서비스의 초기 생성시 '핵심 element' 와 '부가 element' 를 구분하고 '핵심 element'의 구성에 변화가 생겨야 한다면 신규 서비스를 생성하도록 한다. 2. 서비스 오퍼레이션에서 사용되는 메시지 정의의 변화는 부가 element 의 추가/삭제만이 가능하다 3. 한 서비스의 오퍼레이션은 추가/수정만이 가능하다. 4. 서비스의 버전이 증가하면 서비스의 명칭과 endpoint URL 을 변경하며, 이전 버전 서비스의 endpoint URL 의 가용성을 보장해야 한다.

3. 결론

SOA 의 환경에서 서비스는 유연하게 계속된 변화를 겪게 된다. 그래서, SOA 의 서비스를 구축하는 데 사용되는 기술은 이런 지속적인 변화를 감당할 수 있는 능력을 가지고 있어야 한다. 본 논문에서 살펴 보았듯이 웹서비스는 몇 가지의 가이드만 지켜진다면 변화하는 서비스들 속에서 문제없이 잘 활용될 수 있는 SOA 의 구현 기술이다.

참고문헌

[1] JAX-RPC 1.1 : http://java.sun.com/xml/downloads/jaxrpc.html

[2] JAX-WS : http://jcp.org/aboutJava/communityprocess/pfd/jsr224/index.html

[3] JAX-RPC 와 JAX-WS 비교 : http://www-128.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc.html

[4] JAX-RPC 의 DII 와 SEI 방식 : http://www-128.ibm.com/developerworks/library/x-tipjaxrpc/index.html

[5] SAAJ 로 SOAP 메시지 송수신 : http://www-128.ibm.com/developerworks/xml/library/x-jaxmsoap/index.Html

[6] JBI (JSR208) : http://www.jcp.org/en/jsr/detail?id=208

[7] 2006 JavaOne : http://java.sun.com/javaone/sf/2006/index .jsp