제5차 Web Service
유럽 컨퍼런스 국외출장 결과보고

2007. 12.

통 계 정 보 국

목 차

5차 Web Service
국제 컨퍼런스 국외출장 결과보고

Ⅰ. 출장개요

1. 출장목적

□ 서비스 지향 웹 프로그램 등 최신 정보기술에 대한 지식 습득 및 동향파악을 바탕으로 현재 우리 청에서 추진 중인 각종 통계정보 생산 및 서비스 시스템의 효율적 구축방안 모색

□ 정보시스템 관련 선진기술을 비교 연구하여, 향후 우리 청의 효율적 통계정보 서비스 체계 구축에 반영

2. 출장기간 : 2007. 11. 25 ~ 11. 30. (6일간)

3. 출장장소 : 독일 할레

4. 출장자

□ 정보화기획과 과장 진찬우,
정보화기획과 5급 황영자, 전산개발과 5급 김미애

5. 주요 회의내용

□ 최신 웹서비스 기술

□ 서비스 지향 엔지니어링 및 최적화

□ 스토리지 기술 및 데이터센타 환경

□ 웹서비스 보안
□ 그리드/유틸리티 컴퓨팅

Ⅱ. 컨퍼런스 소개

1. 명칭 : ECOWS ’07
(5th IEEE European Conference on Web Services)

2. 컨퍼런스 내용 : Web Service 관련 내용
□ 영국 외 외국논문 참가(이탈리아, 호주, 독일 등)

3. 연혁
□ 제 1차 ECOWS 개최 (2003년) ~ 제 4차 ECOWS (2006년)

4. 5차 ECOWS 조직구성

구 분	성명 및 소속
General Chair	- Wolf Zimmermann (Universitat Halle, Germany)
Programme Co-Chairs	- Birgitta Konig-Ries (Universitat Jena, Germany) - Claus Pahl (Dublin City University, Ireland)
Workshop Chair	- Ronald Maier (Universitat Innsbruck, Austria)
Business Chair	- Rainer Neumann (PTV AG, Germany)
Local Chair	- Romona Vahrenhold (Universitat Halle, Germany)
Web Chair	- Andreas Both (Universitat Halle, Germany)
Steering Committee	- Liang-Jie Zhang (IBM Research, USA) - Abraham Bernstein (Universitat Zurich, Switzerland) - Thomas Gschwind (IBM Research Switzerland)
Program Committee	- Farhad Arbab (CWI, The Netherlands) - Luciano Baresi (Politecnico di Milano, Italy) - Steven Battle (HP Labs, UK) - Boualem Benatallah (University of New South Wales, Australia) - Walter Binder (University of Lugano, Switzerland) - David Breitgand (IBM Research, Israel) - Chris Bussler (BEA System Inc, USA)

Ⅲ. 5차 컨퍼런스 내용(요약)

1. SOA and Web Service : New Technologies, New Standards - New Attacks

Meiko Jesen(독일, Chrian-Albrechts대학)
Nils Gruschka(독일, Chrian-Albrechts대학)
Ralph Herkenhoner(독일, Chrian-Albrechts대학)
Norbert Luttenberger(독일, Chrian-Albrechts대학)

개 요

인터넷 커뮤니케이션의 새로운 패러다임으로 간주되는 웹서비스는 수 많은 기준과 기술을 소개했다. 오랜 네트워크 경험에도 불구하고, 웹서비스는 다른 어떤 오픈 네트워킹 시스템보다 보안공격에 취약하다.
웹서비스는 일반적인 인터넷 프로토콜 공격에 노출되어 있고, 특히 웹서비스만을 목표로 하는 새로운 종류의 공격에도 노출되어 있다.
이런 심각한 공격과 더불어, 대부분의 공격은 공격자의 작은 노력만으로도 아주 쉽게 발생할 수 있다.
이 논문에서는 웹서비스 취약성들을 살펴본다.
이런 위협의 실제성을 증명하기 위해 우리는 몇몇 웹서비스에 대해 샘플공격을 취했다. 마지막으로, 그런 공격의 예방과 공격에 따른 피해를 최소화하기 위한 일반적인 대응책에 대해 알아본다.

I. 서론

웹서비스와 SOA는 지난 십 년 동안의 가장 중요한 기술적 혁신으로 여겨진다.
그럼에도 불구하고, 웹서비스와 SOA의 장점들은 새로운 기술들이 초래한 심각한 결점들과 직면해있다. 그 중 가장 심각한 이슈들은 웹서비스 보안이다.
보안의 일반적인 요구사항들은 무결성(Integrity), 기밀성(Confidentiality), 가용성(Availability)이다.

이런 보안요소들의 침해를 목표로 행해지는 어떤 행동들은 공격이라 부르며, 이런 공격의 가능성은 취약성(Vulnerability)이라 부른다.
이 논문에서 우리는 웹서비스 영역에서 조사기간동안 발견한 보안관련 이슈들을 살펴본다.
이 리스트들은 완벽하지는 않지만, 우리가 시험했던 가장 중요한 공격들의 모음이다.
우리의 연구는 가용성에 초점을 맞춰서 행해졌기 때문에 공격의 대부분은 DoS(Denial-of-Service) 범주에 속해있다.
DoS 공격으로 인한 피해는 매일 뉴스에서 볼 수 있다. 예를 들어, DDoS(Distributed Denial-of-Service)의 에스토니아 정부공격과 2007. 4월 5월의 일반 웹사이트들의 공격이 있다.
이런 공격은 네트워크 레이어의 공격기술인 봇넷(botnet)에 의해 이뤄졌다.
이 논문에서 살펴보겠지만, 웹서비스에서 Dos 공격은 non-웹서비스 시스템에 비해 훨씬 수월하게 일어날 수 있다.
공격들은 다양한 시스템 측면을 포함하고 있다.
보안장치가 전혀 없는 웹서비스에서부터, WS-Security가 가능한 웹서비스, 마지막으로 웹서비스 컴포지션에서의 웹서비스까지 살펴 볼 것이다.
모든 웹서비스 구성요소에 대해 다 살펴봐야 하지만, 우리는 웹서비스의 기준으로 여겨지는 WS-BPEL(줄여서 BPEL로 사용)에 대해서만 살펴 볼 것이다.
논문은 다음과 같이 구성되어 있다.
다음 섹션에서는 웹서비스 보안과 BPEL에 대한 기본적인 개념과 정의에 대해 살펴본다.
섹션 III에서는 웹서비스에 있어서 공격의 취약성들에 대해 살펴본다.
그리고 일반적인 대응책에 대해서 알아보고, 마지막으로 섹션 V에서는 이 논문에서 살펴본 사항들에 대해 결론을 도출한다.

II. 기본적 개념

A. WS-Security

웹서비스 보안의 가장 중요한 요소는 WS-Security이다.
WS-Security는 메시지의 무결성 및 비밀성 보호와 송신자의 인증을 목적으로 단순 객체 접근 프로토콜 SOAP 메시징을 확장한 것이다.
WS-Security는 WS-Security 확장자를 가진 SOAP 헤더블럭(일반적으로 보안헤더라

불러짐)을 가지고 있다. 추가적으로, WS-Security는 XML 서명과 XML 암호화와 같이 XML Security 기준들이 어떻게 SOAP 메시지에 적용되는지를 정의한다.
XML 서명은 무결성을 보장하거나 인증을 증명하기 위해 XML fragment를 디지털로 서명한다. 이 서명된 결과 즉 암호화된 digest는 서명부분에 위치하고 다시 보안헤더에 추가된다.
XML 암호화는XML fragment가 데이터의 기밀성을 보장하기 위해 암호화된다.
암호화된 fragment는 내용으로 암호화된 EncryptedData로 대체된다.
또한, XML 암호화는 키 전송을 목적으로 EncrypedKey를 정의한다.
암호키에 적용되는 암호화는 복합 암호화이다.
XML fragment는 랜덤 추출된 대칭키로 암호화 되고 수신자의 공개키로 암호화된다.
SOAP 메시지에서는 the EncryptedKey(가능하다면)는 보안헤더안에 포함되어 있다.
암호화와 서명에 덧붙여, WS-Security는 안전한 전송을 위해 보안 토큰을 정의한다.
즉, UsernameToken 또는 X.509 인증서이다.
WS-Security의 메커니즘의 중요한 특징은 높은 유연성이다.
WS-Security는 SOAP 메시지의 임의의 어떤 부분에서도 적용될 수 있다(비록 다른 부분은 적용되지 않았지만.)
결과적으로, 웹서비스 서버와 클라이언트는 사용되어질 WS-Security를 정의하기 위한 보안 정책을 결정해야만 한다. 웹서비스 description(WSDL)에서 서버는 필요한 보안 부분을 설명하기 위해 WS-Security Policy document를 사용한다.
WS-Security Policy는 SOAP 메시지의 암호화 또는 서명시 필요한 알고리즘과 필요한 보안토큰을 명세화한다.

B. BPEL

BPEL(The Business Process Execution Language)는 웹서비스 구성요소의 중요한 기준이다. 각 BPEL document는 BPEL process의 workflow를 정의한다.
프로세스들은 BPEL 운영 환경하에서 실행되어져야 할 명령어를 뜻하는 활동들로 이루어져있다. 이런 활동들은 1) 웹서비스의 호출과 관련된 communication 활동과 2) 실행순서를 나타내는 structure 활동들 3) 변수 접근, workflow에서 임시적인 제한 또는 fault handling과 같은 기본적인 활동들로 구분된다.
런타임시, 각 BPEL 프로세스들은 여러 개 process instances를 가질 수 있다.
이렇게 함으로써 같은 프로세스의 동시 접근이 가능하다.
BPEL의 중요한 점은 비동기 커뮤니케이션 방식을 이용하는 것이다.
일반적인 웹서비스 콜은 즉각적인 응답 메시지를 요청하는 요청 메세지들로 구성된다.

요청자는 응답 메시지가 도착하기까지 보류된다.
특별한 언어구조를 사용함으로써 BPEL.은 비동기적 통신이 가능하며, 요청자는 요청 메시지를 보내고도 수행을 계속할 수 있다.
이런 통신방식은 한번의 웹서비스콜의 타임아웃 이내에 수행될 수 없는 비교적 긴 작업에 적합하다.
그런 비동기적 응답은 HTTP 방식으로는 구현이 되지 않는다.
대신 BPEL 엔진은 요청자의 웹서비스를 호출낸다.
콜백을 위해 필요한 명세화는 WS-Addressing 이다. WS-Addressing은 요청자가 요청 메세지내에 임의의 endpoint를 명시하고, BPEL 엔진이 요청자의 웹서비스를 호출하기 위한 모든 정보를 담고 있다.
BPEL 엔진이 수행해야하는 추가적인 작업은 message correlation 이다.
BPEL 엔진은 동시에 하나의 BPEL 프로세스에 대해 여러 개의 인스턴스를 실행할 수 있으므로, target 프로세스의 인스턴스를 구별하기 위해 정해진 메시지 데이터 필드를 사용한다. 이것은 correlation sets라 불려진다.

III. 공격

이 장에서 우리는 웹서비스의 주요 공격에 대해서 알아본다.
각 공격에 대해서 임의의 공격 방법과 그에 따른 충격을 살펴보고, 적절한 시스템 환경하에서 구체적인 공격을 시험해 보았다.
마지막으로, 특별한 공격에 대한 대응책에 대해서도 논의한다.

A. Oversize Payload

DoS(Denial-of-Service) 공격의 중요한 특징은 자원의 소모이다.
Dos 공격은 서비스 호스트 시스템의 자원(메모리, 처리기, 네트웍 bandwidth)을 소모함으로써 서비스 가용성을 불가능하게 하는 것이 목표이다.
그런 자원 소모 공격을 수행하는 고전적인 방법은 아주 큰 요청 메세지를 이용하여 서비스를 요청하는 것이다.
이 공격을 Oversize Payload 공격이라 부른다.
웹서비스에서 Oversize Payload 공격은 XML 처리에 있어서 많은 메모리 소비가 일어나기 때문에 쉽게 일어날 수 있다.
SOAP 메시지 처리에서 사용되는 전체 메모리 사용량은 단지 그 메시지 자체

처리에서 사용되는 메모리 사용량보다 훨씬 크다.
이것은 DOM(Document Order Model)이 트리구조로 펼쳐서 처리되는 XML 프로세싱 모델 때문이다.이 모델을 사용하면 XML document는 읽혀지고, parsing 되고,
메모리로 적재되며 원래 XML document 보다 훨씬 더 많은 메모리 공간을 차지한다.
일반적인 웹서비스 환경하에서, 메모리 사용량이 2에서 30으로 증가했다.

Example: Axis 웹서비스는 큰 SOAP 메시지를 이용하여 공격했다. 웹서비스를 위한 파라미터 값으로 많은 element를 구성했다.

```
<Envelope>
 <Body>
  <getArraylength>
   <item>x</item>
   <item>x</item>
   <item>x</item>
    ...
  </getArraylength>
 </Body>
</Envelope>
```

SOAP 메시지의 총 사이즈는 1.8MB 이다. 이 메시지 처리를 위해서는 1분 이상의 전체 CPU 사용과 50MB 이상의 메모리 사용이 필요했다.
심지어 1.9MB(응답이 없었음) 메시지는 out-of-memory 현상을 초래했다.
Oversize Payload 공격의 확실한 대응책은 요청되는 SOAP 메시지의 총 버퍼사이즈를 제한하는 것이다. 이 경우에는 실제 메시지 사이즈를 체크하고 미리 정의된 한계를 초과하는 어떤 메시지에 대해서 거부하면 된다.
이 방법은 .NET 2.0 프레임워크에서 사용되어졌고 4MB를 초과하는 SOAP 메시지는 디폴트로 무시된다. 이 방법은 구현하기에 매우 간단하지만 웹서비스 메시지에는 적당하지 않다. 좀 더 적당한 방법은 XML infoset에 대해 제한을 가하는 것이다.
이 방법은 웹서비스 description 내에 XML 스키마를 수정하고 요청되는 SOAP 메시지가 이 스키마와 유효한지 체크하는 것이다.
상세한 접근 방법은 섹션 IV에서 다룬다.

B. Coercive Parsing

웹서비스 요청 처리에 있어서 첫번째 단계는 SOAP 메시지를 parsing하고 용프로그램에서 사용할 수 있도록 메시지를 변환하는 것이다. 특별히 namespace를 사용하는 경우 XML은 다른 메시지 코딩방식에 비교해서 파싱 작업이 길고 복잡하다. 그러므로, XML 파싱 프로세스는 Coercive Parsing 공격이라 불리우는 DoS 공격을 받을 가능성이 있다.

Example: 아래의 공격은 an Axis2 웹서비스에 대해서 이뤄졌다. 이 공격은 연속적으로 테그를 오픈하는것이다.

```
<x>
 <x>
  <x>
      ...
```

이 공격의 결과 목표시스템의 CPU 100% 사용을 야기시켰다.
서비스의 가용성은 급격히 줄어들었고, 입력되는 메시지는 마침내 지속적으로 150 byte/s로 전달됐다. 만일 공격이 아주 느린 bandwidth를 목표로 했다면 성공했을것이다.
웹서비스 서버는 그 연결을 중지시키지 않았기 때문에 그 공격은 분명히 끊임없이 지속되었을것이다. 하지만 실험이라 1시간 후에 공격을 멈추었다.
자원 소모를 목표로 하는 전형적인 Coercive Parsing 공격은 수많은 namespace를 선언하거나 지나치게 큰 prefix names 또는 namespace URIs 또는 겹겹히 쌓여진 XML 구조 등을 이용한다.
이런 공격들은 각기 다른 대응책을 필요로 한다.
복잡하거나 nested된 XML documents에 기초를 둔 공격은 스키마 확인 작업을 통해서 막을 수 있다. 잘못 사용된 namespace는 방지하기 훨씬 어렵다.
XML은 XML당 namespce의 숫자나, URI의 namespace의 길이에 제한을 두지 않기 때문에, namespace의 숫자나 길이에 제한을 두는 것은 임의적이고, 의도하지 않은 메시지의 거부를 야기할 수 있다.

C. SOAPAction Spoofing

SOAP 요청에 의한 웹서비스의 실제적인 오퍼레이션은 SOAP body element의 첫번째 child element를 찾아내는 것이다.
그 다음, HTTP 헤더 필드인 "SOAPAction"이 식별을 위해서 사용된다.
비록 이 값이 실제 오퍼레이션을 위한 힌트에 불과하지만, SOAPAction 필드는 종종 요청된 오퍼레이션만을 한정하는 것으로도 사용되어진다.
이것은 HTTP 헤더가 어떠한 XML 프로세싱을 요구하지 않는 bogus 최적화에 바탕을 두고 있다. 이 두 부분의 식별 작업은 두 가지의 공격을 가능하게 한다.
첫번째 공격은 SOAP body 에 명시된 것과 다른 오퍼레이션을 시도하는 것이다..

Example: 다음 공격은 닷넷 웹서비스에서 실시되었다. 실행된 서비스는 두가지 오퍼레이션을 제공한다. 첫번째는 op1(string s) 이고 두번째는 op2(int x) 이다.
다음과 같은 메시지가 서비스에 보내졌다.

```
POST / Service.asmx HTTP/1.1

SOAPActiong:"op2"

<Envelope>
 <Body>
   <op1>
     <s>Hellow</s>
   </op1>
 </Body>
</Envelope>
```

이 메시지로 요청된 작업은 op2(0)이다. 이것은 오퍼레이션이 HTTP 헤더의 SOAPAction 값으로만 선택되어졌다는 것을 나타낸다.
설상가상으로 파라미터의 이름과 타입이 일치하지 않음에도 불구하고 잘못된 오퍼레이션이 수행되었다.
이것은 HTTP 헤더의 수정으로 인해 SOAP 메시지의 creator가 의도하지 않은 메소드가 호출될 수 있음을 나타낸다. HTTP 헤더는 WS-Security로 보안이 되지 않기 때문에 각 SOAP 중개자에 의해 다시 정의되므로 쉽게 수정될 수 있다.

두번째 SOAPAction spoofing 공격은 웹서비스 클라이언트에 의해 수행되고 HTTP 게이트웨이를 우회하려는 것이다.

Example: 다음 공격은 Axis2 웹서비스 환경하에서 실행되었다.
요청된 서비스는 hidden과 visible 오퍼레이션이다.
다음의 메시지가 서비스에 보내졌다.

```
POST /axix2/testService HTTP/1.1

SOAPAction : "Visible"
<Envelope>
 <Body>
   <hidden />
 </Body>
</Envelope>
```

Axis2 서버는 실제로 SOAPAction 값을 무시하고 hidden 오퍼레이션을 수행했다.

SOAPAction Spoofing 공격의 대응책은 SOAP body의 컨텐츠에 의해 오퍼레이션을 수행하는 것이다. 게다가, 오퍼레이션은 HTTP 헤더와 SOAP body에 의해서 결정되어야 하고, 비교해서 어떤 차이가 있다면 위협으로 간주하고 그 웹서비스의 요청을 거부 하는것이다.

D. XML Injection

XML Injection 공격이란 오퍼레이션 파라미터나 XML tag를 포함한 컨텐츠를 삽입함으로써 SOAP 메시지의 구조를 수정하려는 것이다.
그런 공격은 "<" , ">" 과 같은 특별 문자가 사용이 가능한 경우에 일어난다.
웹서비스 서버측면에서는, 이 컨텐츠는 SOAP 메시지의 일부로 간주되어지므로, 뜻하지 않은 효과를 야기할 수 있다.

Example: 다음 공격은 닷넷 웹서비스 환경하에서 실행되었다. 요청된 서비스의 메쏘드는 a,b 파라미터를 사용하고 타입은 int 이다.

```
<Envelope>
 <Body>
   <HellowWorld>
     <a> <b>1</b></a>
     <b>2</b>
   </HellowWorld>
 </Body>
</Envelope>
```

위와 같은 메시지는 XML Injection 공격을 야기했다. 1 컨텐츠가 삽입되었다.
SOAP 메시지가 웹서비스 스키마를 위반하고 있기 때문에 그 서비스는 거부되어져야한다.
그러나 실제에서는 그 메시지는 닷넷에 의해 받아들여졌고 이 요청에 대한 답은 a=1,b=0였다. 이렇게 공격자는 컨텐츠를 수정함으로써 b의 값을 수정할 수 있었다.
제한된 데이터에 대한 접근과 같이 의도하지 않은 결과를 초래할 수 있다는 것은 위의 결과로 쉽게 유추할 수 있다.
공격을 탐지하기 위한 중요한 단계는 SOAP 메시지의 엄격한 스키마의 확인이다. 가능하다면 데이터 타입의 확인까지 필요하다.
위와 같은 확인 작업이 있었다면 샘플공격은 불가능했을 것이다.

E. WSDL Scanning

WSDL(Web Service Description Language)은 파라미터, 데이터 타입, 네트워크 바인딩을 포함한 서비스의 오퍼레이션을 나타낸다. 일반적으로 몇몇 오퍼레이션은 로컬 네트워하에서만 접근이 가능해야하고(internal 오퍼레이션), 반면에 다른 오퍼레이션은 외부 네트워크를 위해 사용되어진다(external 오퍼레이션)

만약 웹서비스가 일반적인 웹서비스 프레임워크를 이용해서 구성되었다면, WSDL은 모든 오퍼레이션을 포함한다. 이 경우에 외부 클라이언트는 internal 오퍼레이션에 대한 정보도 획득할 수 있고 심지어 internal 오퍼레이션을 호출할 수 있다.
그런 접근을 피할수 있는 첫 번째 단계는 외부 클라이언트는 external 오퍼레이션만 접근할 수 있도록 WSDL을 분리하는 것이다.

그러나, 웹서비스 endpoint가 외부에서 접근이 가능하다면 공격자는 생략된 오퍼레이션을 임의로 추측하고 그 오퍼레이션 콜을 시도할 수 있다.
이런 공격은 WSDL Scanning이라 부른다.

예를 들어, 인터넷 shop 시스템은 고객의 주문을 처리하는 sendOrder 메쏘드와 그 주문을 처리하는 admonOrders가 필요하다.물론 주문을 처리하는 admonOrders는 그 shop 내부에서만 call되도록 되어있다.만일 sendOrder와 adminOrder가 웹서비스에 모두 제공되어있다면 공격자는 sendOrder에 대한 정보만을 가지고 admin 메쏘드에 대한 정보도 쉽게 찾을 수 있다.
internal 오퍼레이션에 대한 취약한 예는 레거시나 디버그 메쏘드 등이 있다.
이 공격에 대한 대응책은 웹서비스에서 internal과 external을 분리하고 가능하다면 서버도 분리하는 것이다.
만일 이것이 불가능하다면, 외부 클라이언트에 의한 internal 오퍼레이션에 대한 호출은 통제되어야 하고 거부되어야 한다.
이것은 boder 게이트웨이의 역할이다.
아쉽게도 external 과 internal 요청 메시지는 똑같은 IP 주소, TCP 포트, 심지어 HTTP URL 까지 같다. 위와 같이, filter와 HTTP firewall은 웹서비스 오퍼레이션을 받아들여야할지를 결정할 수 없다. 그러므로, Web-Service-aware XML firewall이 필요하다.

F. Metadata Spoofing

웹서비스 클라이언트는 웹서비스 서버에서 제공하는 메타데이터 documents로부터 웹서비스 요청(메세지 포맷, 네트워크 위치, 보안 요구사항 등)을 위해 필요한 모든 정보를 검색한다.
실제로 이런 메타데이터는 HTTP나 mail과 같은 커뮤니케이션 프로토콜을 통해 공급된다. 이런 환경은 Metadata Spoofing이 가능한 새로운 공격의 가능성을 제공한다.
이 범주에 속하는 가장 일반적인 공격의 형태는 WSDL Spoofing 과 Security Policy Spoofing 이다. WDSL Spoofing의 일반적인 형태는 네트워크 endpoint에 대한 수정과 security policy의 불법적인 reference이다.
endpoint의 수정은 공격자로 하여금 몰래 정보를 엿보거나 데이터 수정을 가능하게 한다.

이런 Metadata Spoofing 공격을 피하기 위해서는, 모든 메타데이터 documents는 사용자의 신원을 확인하는 등 인증 작업을 거쳐야 한다.
그러나 SOAP 메시지의 보안 메쏘드에 비해 메타데이터의 보안 메커니즘은 표준화되어 있지 않다.

G. Attack Obfuscation

웹서비스에 WS-Security를 사용하는 것은 서비스 가용성에 있어서 새로운 문제를 야기한다. 민감한 데이터에 대해서 인증을 함으로써, XML 암호화는 메시지 컨텐츠를 감춘다.
이런 암호화된 컨텐츠는 의도화된 공격-Oversize Payload, Coercive Parsing, XML Injection-을 포함할 수 있기 때문에, 암화화는 공격을 숨길 목적으로 사용되어질 수 있다.
이런 공격의 가장 중요한 문제점을 공격을 알아채기 힘들다는 것이다.
스키마 확인과 같이 메시지 구조를 파악하기 위해서는 복호화가 필요하다.
여기에 목표시스템이 영향을 받을 수 있는 가능성이 두 가지가 있다.
만약 복호화가 메시지 확인후에 일어난다면, 악의를 가진 메시지 켄텐츠는 메시지 확인작업을 무사히 통과하게 된다.
만약 복호화가 메시지 확인전에 일어난다면, 시스템은 XML과 암호관련 처리 때문에 메시지 복호화하는 동안 다른 작업을 할 수 없게 된다.
그러므로, obfuscated 공격은 목표시스템에 영향을 미치게 된다.

Example: Axis2 obfuscated Oversize Payload 공격에서 약점이 발견되었다.
이 공격 시나리오에서, 하나의 SOAP 메시지가 서버에 요청되었다.
암호화되고 서명된 1MB의 메시지를 보냄으로써 23초 동안의 CPU full과 out-of-memory exception이 일어났다. 반면에 Java runtime 환경하에서는 추가로 90MB의 시스템 메모리가 필요했다.
반면에, 암호화되지 않은 20MB 사이즈의 메시지는 Axis2 서버에서 1초도 안되는 처리시간이 필요했다.
이 공격을 예방하기 위해서는 해독된 컨텐츠에서 메시지의 확인 작업을 거치는 것이다.
가장 좋은 방법은 순차적으로 복호화와 확인 작업을 거치는것이다.
이 방법은 메모리 소비를 줄이고 악의성이 있는 메시지의 조기 발견을 가능하게 한다.

H. Oversized Cryptography.

WS-Security가 야기한 또 다른 문제는 보안 요소의 유연한 사용이다.
암호화는 SOAP 메시지의 어떤 부분에서도 사용될 수 있다. 보안헤더의 유연한 layout은 엄격한 스키마의 확인을 불가능하게 한다. 보안 elements를 사용하는 다양한 가능성은 각 element의 스키마 확인 작업을 제한하고, order나 occurrence 체크도 불가능하다.
이러한 유연성이 공격에 악용될 수 있다.
self-evident 공격은 지나치게 큰 보안헤더에서 발생할 수 있다.
만약 목표시스템이 전체 보안헤더를 처리하거나 buffering 한다면, 목표시스템은 Oversize Payload 공격에서 살펴 본 것처럼 똑같은 방식으로 영향을 받게 된다.
지나치게 큰 보안헤더를 가진 좀 더 복잡한 유형의 공격은 연속된 암호키를 이용하는것이다.
이와 같은 체인에서, 각 암호키는 다음 키를 암호화하기 위해 사용되어 진다.
이렇게 목표시스템은 각 암호화키를 복호화 해야 하고, 각 키는 다음의 복호화를 위해 또 다시 필요해진다.
이 것은 목표시스템을 두가지 측면에서 영향을 미친다.
첫째, 목표시스템은 각 키를 buffer 해야만 한다. 왜냐면 이 메시지의 처리가 끝날때까지 알 수 없으므로(암호화된 키가 다른 컨텐츠를 위해 사용되었는지 확인해야함)
이것은 많은 메모리 소비를 요구한다.
두 번째, 복호화는 많은 리소스를 필요로 한다.
특별히 일반적으로 키 전송을 위해서 사용되는 비대칭 알고리즘에서는 아주 많은 CPU를 소모하게 된다.
이런 공격은 SOAP 메시지에서 겹겹이 쌓여진 마치 (러시안 matryoshka 인형처럼) 수많은 암호화 블록을 사용한다. 목표시스템은 모든 nested block을 각 inner content로 처리하기 위해 복호화를 해야만 한다.
이 작업은 수 많은 복호화로 인한 많은 CPU load를 야기한다.
또한, 만약 복호화된 컨텐츠가 다음 처리를 위해 buffering 되어야 한다면, 메모리 소비는 원래 메시지 사이즈의 몇배가 될 것이다.
대응책으로서 WS-Security elements의 사용은 제한되어져야 하고, 한계를 초과하는 메시지는 거부되어야 한다.

I. WS-Addressing Spoofing

비동기적 웹서비스를 위한 WS-Addressing은 수 많은 공격의 가능성을 가지고 있다. 가장 간단한 방법은 callback endpoint reference로 임의의 유효하지 않은 endpoint URL을 사용하는 것이다.
결과적으로 BPEL 엔진은 관련된 프로세스를 처리하고 요청자에게 callback 시도한다. 이것은 endpoint URL 때문에 바로 에러를 야기하거나(connection 거절, HTTP 서버 에러 또는 SOAP fault) 타임아웃을 초래한다.
이렇게 BPEL 엔진은 execution fault나 matching fault handlers를 호출하게 된다. 결과적으로 BPEL 엔진은 전체 프로세스를 수행하게 되고 그 다음 rollback을 수행하게 된다. Flooding 공격에서 처럼, 이것 또한 BPEL 엔진의 부담을 야기한다. 앞에서 살펴본 flooding 공격과 비교해볼때, 이 공격에 의한 workload는 가장 크다.
WS-Addressing spoofing 공격에 대한 가장 확실한 대응책은 요청자의 endpoint URL을 확인하는 것이다. 이것은 메시지의 빠른 거부를 가능하게하고, BPEL 엔진의 불필요한 workload를 방지한다.

J. Workflow Engine Hijacking

이 공격은 WS-Addressing spoofing을 다시 사용하는 것이다.
공격자의 endpoint URL은 목표시스템에 존재하는 어떤곳을 가리킨다.
결과적으로 BPEL 엔진은 계속해서 공격자의 요청에 대해 응답을 하게 될 것이다.
따라서, 이 공격하에서 서비스는 유효하지 않은 SOAP 메시지를 가진 아주 큰 메시지를 받게된다.(더욱 나쁜 것은 유효한 메시지일 수 도 있다.)
BPEL 엔진은 아주 강력한 서버 환경에서 주로 작동하므로, 목표시스템은 DoS로 공격을 받을 수 있다.
이렇게 공격자는 워크플로우 엔진 시스템을 이용하여 목표시스템을 다운시킬 수 있다. indirect flooding 공격과 더불어, BPeL 엔진은 인터넷 방화벽안에 있는 어떤 시스템의 공격을 위한 매개체로 사용될 수 있다.
일반적인 indirect flooding 공격과 달리, WS-Addressing을 사용하면 공격자 자신이 목표시스템을 스스로 선택할 수 있게된다.
이 공격의 대응책은 WS-Addressing Spoofing 공격의 대응책과 일반적으로 같다.

IV. 일반적인 대응책

다른 시스템과 마찬가지로 웹서비스 공격은 수 많은 취약성에 기초한다.
그러므로 공격에 대한 대응책은 아주 광범위하다.
그럼에도 불고하고 일반적인 방어 메커니즘이 존재한다.

A. Schema Validation

스키마 확인은 웹서비스 description과 일치하지 않는 메시지를 이용하는 공격에 대응하는 방법이다.그런 공격은 메시지 syntax의 이탈이라고 불려진다.
WSDL로 생성된 XML 스키마와 요청된 메시지를 확인함으로써 III-B and III-D 섹션에서 살펴 본 것처럼 공격이 감지될 수 있다.
그럼에도 불구하고 현재의 웹서비스 프레임워크에서는 스키마 확인은 사용되지 않거나 디폴트로 적용되지 않고 있다.
이것은 주로 퍼포먼스 측면때문이다. 왜냐하면 스키마 확인은 CUP load와 많은 메모리의 사용을 필요로 한다.
스키마 확인은 유효하지 않은 메시지를 이용하는 . SQL Injection 이나 or Parameter Tampering 공격에는 아주 효과적이다.
더군다나 스키마 확인은 다른 대응책을 위한 아주 기초적인 작업으로 사용될 수 있다.
중요한 보기는 III-A 에서 살펴본것처럼, 메시지 처리에 필요한 메모리의 사용을 제한하는 XML infoset의 제한이다.
이것을 소위 Schema Hardening이라 부른다.

B. Schema Hardening.

스키마를 분석하는 일반적인 방법은 복잡한 XML tree를 구조화하는 것이다.
이런 구조화는 몇 개의 boundarie로 수정된다.
예를 들어 만일 웹서비스 description이 수 많은 element로 되어 있다면 리스트는 몇 개의 유한한 element로 대체될 수 있다. XML 스키마에서 element maxOccurs="unbounded는 element maxOcuurs="n"로 대체된다.
대부분의 서비스에서 그런 제한은 쉽게 정의된다.
네트워크 버퍼사이즈의 제한과 비교해볼 때 이런 제한의 잇점은 웹서비스 description에 포함할 수 있으므로 클라이언트가 미리 알 수 있다.
schema hardening의 두번째 잇점은 웹서비스 description에 스키마에서 non-public 오

퍼레이션을 없앨 수 있다.
Web Service description의 수 많은 schema hardening 방법이 있다.
자세한 사항은 12에서 살펴보았다

C. Stirct WS-SecurityPolicy Enforcement

WS-SecurityPolicy는 정책을 만족시키기 위해 SOAP 메시지에 포함되어야 하는 최소한의 security tokens을 정의한다.
앞에서 살펴 본 것처럼, 공격자는 수없이 많은 token을 추가하거나, 목표시스템의 암호화 관련 처리에 많은 메모리 소비를 야기할 수 있다.
이것을 피하기 위해서는 WS-SecurityPolicy document는 최소한의 요구사항과 최대한의 요구사항을 모두 만족해야 한다.
이것은 SOAP 메시지는 security policy에 언급된(더도 말고 덜도 말고) 바로 그 만큼의 security token을 가지고 있어야 한다.

D. Event-based SOAP message processing.

위에서 살펴 본 대응책의 효과성은 실제 이행에 달려있다.
SOAP 메시지와 메시지 스키마의 일치성을 체크하고, 보안정책은 XML과 WS-Security 프로세싱을 필요로 한다. 이런 오퍼레이션은 적당한 리소스를 가지고 구현되어야 하며, 그렇지 않으면 웹서비스 시스템은 취약하게 된다.
커다란 SOAP 메시지를 사용하는 공격은 DOM과 같은 트리구조의 실행환경을 보호시스템으로는 부적합하게 만든다.
그런 이행구조는 메시지 전체가 네트워크로부터 읽어지고 SOAP 메시지가 추가적인 처리 이전에 document 트리로 변환되어져야 한다.
그러므로 공격에 대한 점검이 진행되기 전에, 많은 메모리 사용이 이미 필요하다.
몇 몇 트리구조 이행구조는 document의 일부분으로 구성되고, 메모리 소비를 약간 줄일 수 있지만 트리구조 접근의 근본적인 문제를 해결하지는 못한다. 각 XML document는 모드 한꺼번에 읽혀지고 저장되어야 한다.
보안 시스템은 SAX와 같이 event 구조로 XML을 처리방식을 도입해야 한다.
이 event 기반의 XML 처리방식의 잇점은 유효하지 않은 컨텐츠의 검출의 가능성을 높이고 추가적인 처리를 중지할 수 있다는 것이다.
이 방법으로 메모리와 CPU 사용이 줄어들 수 있다.

스키마 확인 작업은 event 기반의 방식으로 처리될 수 있지만 WS-Security는 아직도 XML tree 방식으로 처리되고 있다.
WS-Security-enabled 메시지는 WS-Security tokens 사이에 수 많은 reference를 포함하고 있기 때문에 event 기반은 실제 구현하기가 어렵다.
그러나, 약간의 제한을 감안하더라도, WS-Security는 event 기반으로 처리될 수 있다.

E. WS-Security

WS-Security에 대한 일반적인 오해는 그 자체가 웹서비스 전체에 대해 자동적으로 보안을 보장하는 것으로 간주하는 것이다.
위에서 살펴본 바와 같이, WS-Security는 웹서비스 메시지에 대한 무결성과 기밀성 보장하는 메커니즘을 정의하고 있다.
그러나, 만약 적절한 WS-Security 정책이 정의되지 않으면 무결성과 기밀성에 대한 공격은 소위 XML rewrite 공격으로 가능해진다.
이 논문에서 보다 중요한 것은, WS-Security는 DoS와 같은 공격에 대항하는 어떤 직접적인 대응책을 갖고 있지 못하다는 것이다.
잘 알려진 가용성을 위한 메카니즘은 접근 통제이다.
접근 통제란 trusted 사용자에게만 서비스에 대한 접근을 제한하는 것이다.
추가적으로 접근통제는 공격자를 배제하거나 공격자에 권한을 요구함으로써 책임성을 가지게 한다.
WS-Security는 인증을 위한 security token을 정의하고, 접근통제를 위해서 사용되어 진다.물론 접근통제는 공격의 위험을 완전히 없애지는 못한다.
심지어 믿을 수 있는 커뮤니케이션 파트너(의도적이든 의도적이지 않던)도 공격을 감행할 수 있다.
게다가, 인증은 키 기반 구조를 필요로 하기 때문에, B2C에는 적합하지 않다.
앞에서 살펴본바와 같이 WS-Security는 새로운 종류의 Dos 공격을 가능하게 한다.
요약하자면, WS-Security는 공격을 막기위한 중요한 수단이지만, 네트워크 위협을 막아주는 매직카펫은 아니다.

V. 결론

이 논문에서 보여진 공격들은 웹서비스와 웹서비스 구성요소의 높은 취약성을 보여줬다.
몇몇 취약성은 수행시 결점으로 인해 나타나지만 반면에 취약성의 대부분은 근본적인

프로토콜의 결점을 가지고 있었다. 기본적인 공격 방법론들은 이런 취약성을 악용하는 것(즉, 대부분 서비스의 가용성이나 서비스 질 자체를 목표로 함)으로 설명했다.
공격은 단순한 웹서비스, WS-Security가 가능한 서비스, 그리고 서비스 기반의 워크플로우 엔진까지 포함한다.
각각의 대응책들은 비로소 공격을 피하기 위한 여러가지 메커니즘으로 결합되어지고 있다.
이런 대응책들은 구현되어져 왔고, 자체 방어 능력을 증명해줬다.
웹서비스 보안 영역에서 진행된 우리의 연구중 일부분은 개념의 확장 및 대응책 실행을 위해 계속 연구를 할 것이다.
우리의 주요 관심사는 WS-Addressing 스푸핑 공격에 대한 대응책이다

2. A Light-weight Framework for Hosting Web Services on Mobile Devices

김연석, 이경호(연세대학교)

요 약

불안정한 연결성 및 빈번한 컨텍스트 변화 등은 모바일 환경에서 웹 서비스를 제공하는데 많은 문제점을 야기시킨다. 이를 해결하기 위해서는 사용자에게 지속적으로 서비스 제공이 가능한 디바이스로의 웹 서비스 이동이 필수적이다. 그러나 모바일 디바이스에서의 웹 서비스 이동, 복제, 그리고 호스팅에 관한 연구는 아직 초기단계에 불과하다. 본 논문에서는 모바일 환경에서의 웹 서비스 이동을 위하여 경량의 웹 서비스 프레임웍과 함께 효과적인 이동정책 수립기법을 제안한다. 제안된 방법은 WS-Policy를 기반으로 다양한 컨텍스트 정보를 기술할 수 있는 스키마를 정의하고, 이를 사용하여 이동정책을 수립한다. 특히, 제안된 스키마는 서비스 제공자의 요구사항을 자유롭게 표현하기 위하여 확장 가능하도록 설계되었다. 이동정책의 유효성을 검증하기 위하여, 제안된 방법을 모바일 웹 서비스 프레임웍에 적용한 결과, 웹 서비스 이동을 통한 연속적인 서비스의 제공이 가능함을 보였다.

I. 서론

웹 서비스(Web Services)는 인터넷과 같이 공개적인 네트워크 및 관련 표준기술을 통해 기존의 애플리케이션을 운영체제 및 프로그래밍 언어에 상관없이 상호운용이 가능하도록 해주는 표준화된 소프트웨어 기술이다. 이러한 웹 서비스는 다양한 디바이스가 혼재되어 있는 유비쿼터스 환경에서 사용자가 필요로 하는 서비스의 제공을 가능하게 함으로서 유비쿼터스 환경을 위한 필수적인 소프트웨어 기술로 인식되고 있다. 한편, 기존 웹 서비스의 대부분은 데스크탑 서버 및 유선 환경을 고려하여 개발되었기 때문에 무선 환경에 적용하기에는 문제가 많다. 그러나 머지않아 모바일 디바이스는 단순한 클라이언트 디바이스가 아닌 하나의 독립적 서비스를 제공하는 강력한 서버 형태로 변모되어 갈 것이다. 즉, 모바일 디바이스는 사용자가 원하는 서비스를 제공하는 서비스 제공자의 역할을 수행할 것이다.

모바일 디바이스를 사용하여 웹 서비스를 제공할 경우 네트워크의 불안정한 연결성 및 빈번한 컨텍스트 변화 등은 지속적인 서비스 제공을 어렵게 만든다. 예를 들어, 서비스 제공자가 서비스를 더 이상 제공할 수 없는 위치로 이동할 경우 사용자는 더 이상 서비스를 받을 수 없다. 그러나 해당 서비스를 적절한 디바이스로 이동시킨다면 사용자는 한결같은(seamless) 서비스를 제공받을 수 있을 것이다. 즉, 모바일 환경에서 사용자에게 한결같은 서비스를 제공하기 위해서는 웹 서비스의 이동이 필수적이다.

기존에 컨텍스트 기반의 웹 서비스 이동에 관한 연구가 다양하게 진행되었다. 그러나 기존 연구의 대부분은 모바일 환경을 대상으로 하지 않거나 사용자의 개입이 필요하다는 단점을 가진다. 뿐만 아니라 모바일 환경에서의 웹 서비스 이동, 복제, 그리고 호스팅에 관한 연구는 아직 초기단계에 불과하다.

Hao 등 은 실시간 어플리케이션의 성능 향상을 위하여 웹 서비스 이동 및 실행을 지원하는 weblet 인프라와 비용모델을 제안한다. 이 방법은 이동 가능한 경량의 웹 서비스를 정의하고 웹 서비스의 실행을 지원하는 weblet 인프라를 구축한다. 특히, 웹 서비스의 타겟 호스트를 결정하기 위하여 유전자 알고리즘(Genetic Algorithm)을 적용한다. 그러나 이동시점의 결정을 위해서 사용자의 개입을 필요로 한다. Pratistha 등은 웹 서비스 이동을 지원하는 프레임웍인 Fluid와 함께 이동 호스트의 결정을 위한 비용모델 모듈(Cost Model Module)을 제안한다. 이 방법은 다양한 비용모델 모듈에서 서비스 제공자가 이동에 필요한 모듈을 선택하면, 이후 웹 서비스 이동이 결정되었을 때 선정된 비용모델 모듈에 의거, 가장 적절한 호스트로 서비스를 이동시킨다. 그러나 제안된 방법은 모바일 환경을 고려하지 않으며, 서비스 제공자가 직접 비용모델 모듈을 선택해야만 하는 단점이 있다. Hemmati 등은 서비스 코드와 실행 상태의 이동을 지원하는 서비스 이동 프레임웍을 제안한다. 제안된 프레임웍은 컨텍스트 매니저에 의해 수집된 컨텍스트 정보를 기반으로 적절한 호스트로의 이동을 결정한다. 특히, 서비스 이동의 효율은 소스 호스트에서 서비스가 실행된 정도에 따라, 그리고 목적 호스트에서 서비스를 재실행 했을 때의 효율에 따라 그 정도가 달라진다. 그러나 이 방법은 웹 서비스를 대상으로 하지 않는다. Riva 등은 애드 혹 네트워크에서의 동적인 컨텍스트 변화를 지원하는 모바일 서비스 이동 프레임웍을 제안한다. 이 방법은 서비스 요청자의 컨텍스트를 모니터링 하여 모바일 서비스를 적절한 노드로 이동시킨 다음 서비스를 제공한다. 그러나 제안된 프레임웍은 사용자의 요청에 의한 컨텍스트의 모니터링이 필요하다는 단점을 갖는다. Corradini 등은 모바일 코드의 이동을 위한 프레임웍을 제안

한다.

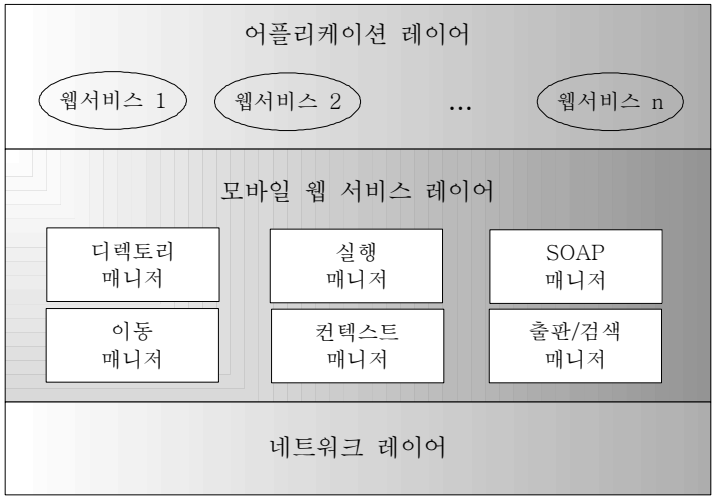
본 논문에서는 서비스의 효과적인 제공을 위하여 웹 서비스 이동 및 복제를 지원하는 모바일 웹 서비스 프레임웍과 함께 언제, 어디로 이동할 것인지를 결정하기 위한 이동정책 수립기법을 제안한다. 제안된 프레임웍은 모바일 환경을 위하여 경량화된 구조를 가지며, 6개의 매니저, 즉, 디렉토리 매니저, 실행 매니저, SOAP 매니저, 이동 매니저, 컨텍스트 매니저 그리고 출판 및 검색 매니저로 구성된다. 한편, 제안된 방법은 이동정책 수립을 위하여 스키마를 정의한다. 제안된 스키마는 서비스 제공자의 요구사항을 자유롭게 표현하기 위하여 UAProf을 포함한 CC/PP 프로파일 등을 적용할 수 있도록 확장 가능하게 설계되었다. 뿐만 아니라 이동시점 및 타겟 호스트의 결정을 위하여 컨텍스트에 따른 우선순위를 표현할 수 있다.

제안된 방법은 스키마를 기반으로 서비스 제공자로부터 필요한 정보를 입력받아 이동정책을 수립한다. 이후, 모바일 웹 서비스 프레임웍은 이동정책을 기반으로 컨텍스트를 모니터링하고, 그 결과에 따라 웹 서비스를 이동시킨 후 서비스를 재실행한다. 이동정책의 유효성을 검증하기 위하여 모바일 웹 서비스 프레임웍에 제안된 방법을 적용한 결과, 웹 서비스 이동을 통한 연속적인 서비스를 제공이 가능함을 보였다.

본 논문의 구성은 다음과 같다. 2절에서는 웹 서비스 프레임웍을 6개의 모듈로 구분하여 설명하고, 이에 기반한 웹 서비스 호출 및 응답 절차에 대하여 간략히 기술한다. 3절에서는 본 논문에서 제안한 스키마 및 이동정책 생성을 위한 사용자 인터페이스에 대하여 자세히 기술하고, 4절에서는 제안된 이동정책 생성기법의 유효성 검증을 통하여 모바일 환경에서의 지속적인 서비스 제공이 가능함을 보인다. 마지막으로 5절에서는 결론 및 향후 연구방향을 기술한다.

II. 모바일 웹 서비스 프레임웍

본 절에서는 웹 서비스 이동을 지원하는 모바일 웹 서비스 프레임웍에 대하여 간략히 기술한다. 제안된 프레임웍은 [그림 1]과 같이 6개의 매니저로 구성된다.



[그림 1] 모바일 웹 서비스 프레임웍

2.1 프레임웍의 구성

제안된 모바일 웹 서비스 프레임웍은 다음과 같이 6개의 매니저, 즉, 디렉토리 매니저, 실행 매니저, SOAP 매니저, 이동 매니저, 컨텍스트 매니저 그리고 출판 및 검색 매니저로 구성된다.

- 디렉토리 매니저

디렉토리 매니저는 모바일 디바이스에 존재하는 웹 서비스를 관리한다. 모바일 디바이스는 그 특성상 다양한 제약조건 즉, 프로세스, 메모리 용량, 배터리 지속시간 등을 갖기 때문에 리소스를 효율적으로 사용해야만 한다. 이를 위하여 디렉토리 매니저는 웹 서비스를 액티브(Active) 모드와 디액티브(Deactive) 모드로 구분한 후 액티브 모드의 웹 서비스만을 사용하도록 설정한다. 또한 애드-혹 환경에서의 웹 서비스 제공을 위하여 WSDL 문서를 저장하고 서비스 요청 시 이를 제공하는 역할을 한다.

- 실행 매니저

모바일 환경에서의 웹 서비스 실행은 유선 환경에서의 그것과 유사하다. 우선, 서비스 제공자가 사용자로부터 요청 메시지를 받으면 실행 매니저는 디렉토리 매니저에 있는 WSDL 문서를 검색한 후 이에 기반하여 사용자의 입력을 요구한다. 만약, 사용자가 적절한 입력을 제공하면 실행 매니저는 요청된 웹 서비스의 인스턴스를 생성한 후 해당 함수를 호출함으로써 서비스를 실행한다.

- SOAP 매니저

SOAP 매니저는 서비스 제공자와 사용자간의 통신을 위하여 SOAP 메시지의 생성 및 파싱을 담당한다. 기존에 모바일 환경을 위한 다양한 SOAP 툴킷이 개발되어 왔다. WSDL 파서와 stub/skeleton 컴파일러를 포함한 C++용 gSOAP을 비롯하여, J2ME 디바이스를 위한 오픈소스 API인 kSOAP 등이 그 예이다. 그러나 kSOAP은 WSDL 문서로부터 사용자 stub을 생성할 수 없다는 단점을 가진다. 한편, 제안된 SOAP 매니저는 편의를 위하여 윈도우 기반이며, 경량의 SOAP 툴킷인 PocketSOAP 툴킷을 사용한다. PocketSOAP의 용량은 약 680K에 불과하다.

- 이동 매니저

모바일 환경에서의 연속적인 서비스 제공을 위해서는 웹 서비스의 이동이 필수적이다. 웹 서비스의 이동은 서비스 제공자의 요청에 의하거나 혹은 디바이스의 배터리가 더 이상 지속될 수 없을 때, 서비스가 제공될 수 없는 위치로 디바이스가 이동했을 때 등 다양한 컨텍스트 변화에 의거 결정된다. 이때, 이동 매니저는 웹 서비스의 코드와 인스턴스 그리고, 디렉토리 매니저에 존재하는 WSDL 문서 등을 적절한 디바이스로 이동시킨다. 특히, 이동 매니저는 제안된 이동정책을 사용하여 이동 가능한 호스트 중 가장 비용이 낮은 호스트를 타겟 호스트로 결정한다.

- 컨텍스트 매니저

제안된 방법은 웹 서비스가 이동 할 타겟 호스트의 결정을 위하여 이웃한 호스트로부터 필요한 컨텍스트를 수집하고 관리한다. 예를 들어, 서비스 제공자가 웹 서비스의 이동을 요청하면, 컨텍스트 매니저는 네트워크를 구성하는 이웃한 호스트에 대하여 필요한 컨텍스트 정보를 요청하고 수집한다. 수집된 컨텍스트 정보는 해당 웹 서비스의 이동정책에 적용되어 타겟 호스트를 결정하는 데 사용된다. 한편, 컨텍스트 매니저는 주기적인 컨텍스트의 갱신을 통하여 네트워크의 트래픽을 감소시킨다.

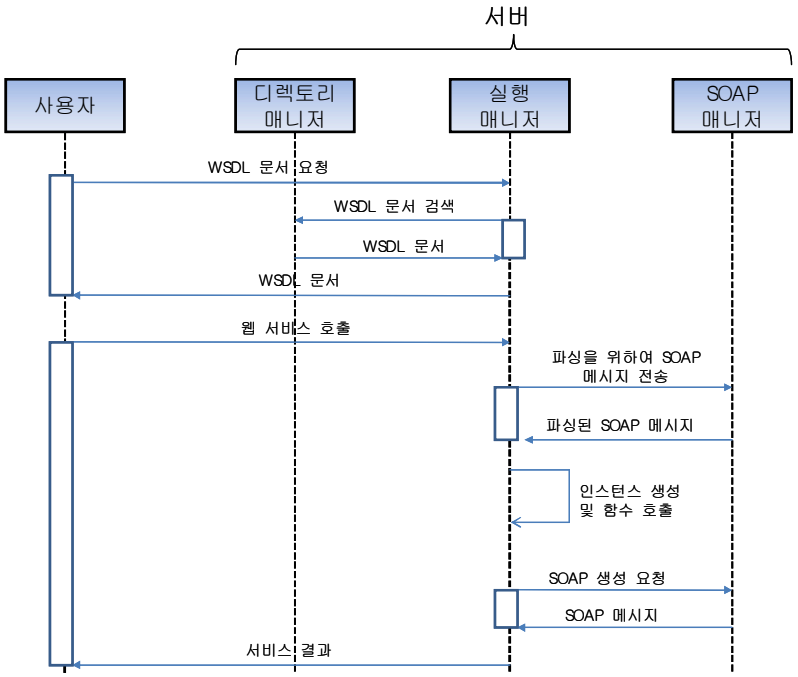
- 출판/검색 매니저

일반적인 의미의 웹 서비스에서, 웹 서비스 제공자는 해당 서비스를 설명하는 WSDL 문서를 브로커에 출판하고, 사용자는 브로커와의 통신을 통하여 서비스를 검색한다. 그러나 모바일 환경 특히, 애드-혹 환경에서의 서비스 제공자 및 사용자는 HTTP 프로토

콜을 사용할 수 없다. 따라서 웹 서비스를 출판하고 검색하는 새로운 방법이 필요하다. 제안된 프레임웍은 출판 및 검색 매니저를 통하여 웹 서비스를 출판하고 검색한다. 출판 및 검색 매니저는 모바일 디바이스에 있는 액티브 모드의 웹 서비스를 출판하는데 특히, 일반적인 웹 서비스의 출판과는 달리 서비스의 이름만을 출판하기 때문에 웹 서비스는 자신의 기능을 표현할 수 있는 이름을 사용해야만 한다.

2.2 웹 서비스 호출 및 응답절차

제안된 프레임웍에 기반한 웹 서비스의 호출 및 응답절차는 [그림 2]와 같이 유선환경에서의 그것과 유사하다. 우선, 사용자가 자신의 디렉토리 매니저를 통하여 원하는 서비스를 검색했다면 해당 호스트에게 웹 서비스의 WSDL 문서를 요청한다. 서비스 제공자의 실행 매니저는 사용자의 요청 메시지를 받은 후 자신의 디렉토리 매니저로부터 WSDL 문서를 검색하고 다시 사용자의 입력을 위하여 해당 문서를 사용자에게 재전송하게 된다. 만약 사용자가 적절한 입력 값을 제공하여 서비스를 호출하면, 서비스 제공자의 실행 매니저는 SOAP 매니저를 통하여 SOAP 메시지를 파싱하고, 호출된 웹 서비스의 인스턴스를 생성한 후 사용자가 원하는 서비스를 실행한다. 이후 그 결과를 다시 SOAP 매니저를 통하여 SOAP 메시지로 변환한 후 사용자에게 적절한 정보를 제공하게 된다. 한편, 사용자의 SOAP 매니저는 수신된 결과를 파싱한 후 적절한 인터페이스를 통하여 사용자에게 정보를 제공한다.



[그림 2] 웹 서비스 호출 및 응답 절차

III. 이동정책 수립기법

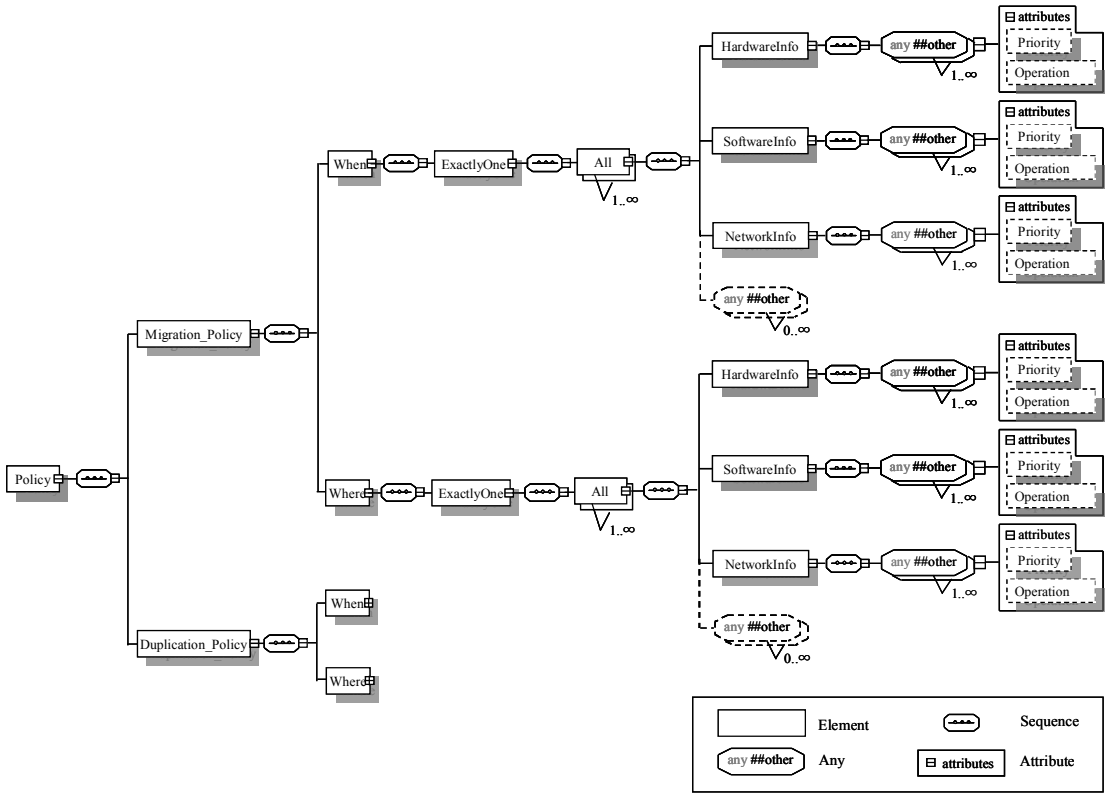
웹 서비스의 이동을 위해서는 해당 웹 서비스가 언제, 어디로 이동해야 할 것인지를 명세한 이동정책이 필수적이다. 이를 위하여 제안된 방법은 W3C의 표준인 WS-Policy에 기반하여 이동정책 수립기법을 제안한다. WS-Policy란 웹 서비스의 WSDL 문서가 기술하지 못하는 다양한 명세 즉, 서비스 요구 사항, 선호도, 기능 등의 서술이 가능한 표준으로서, WSDL 문서에 쉽게 귀속될 수 있다는 장점을 가진다. 또한 하나의 서비스에 대하여 다양한 정책을 수립할 수 있기 때문에 컨텍스트 및 상태 변화에 따른 적절한 대응이 가능하다. 제안된 방법은 [그림 3]과 같이 웹 서비스의 이동정책 수립을 위한 스키마를 정의하며, 웹 서비스 제공자로부터 최소한의 정보를 입력받아 이동정책을 수립한다.

3.1 제안된 스키마 및 사용자 인터페이스

제안된 스키마는 크게 이동 및 복제를 위한 "Migration_Policy"와 "Replication_Policy" 엘리먼트로 구성된다. 이는 이동 및 복제정책을 구분함으로써 상황에 따른 적절한 대응이 가능하게 하기 위함이다. 한편, "Migration_Policy"과

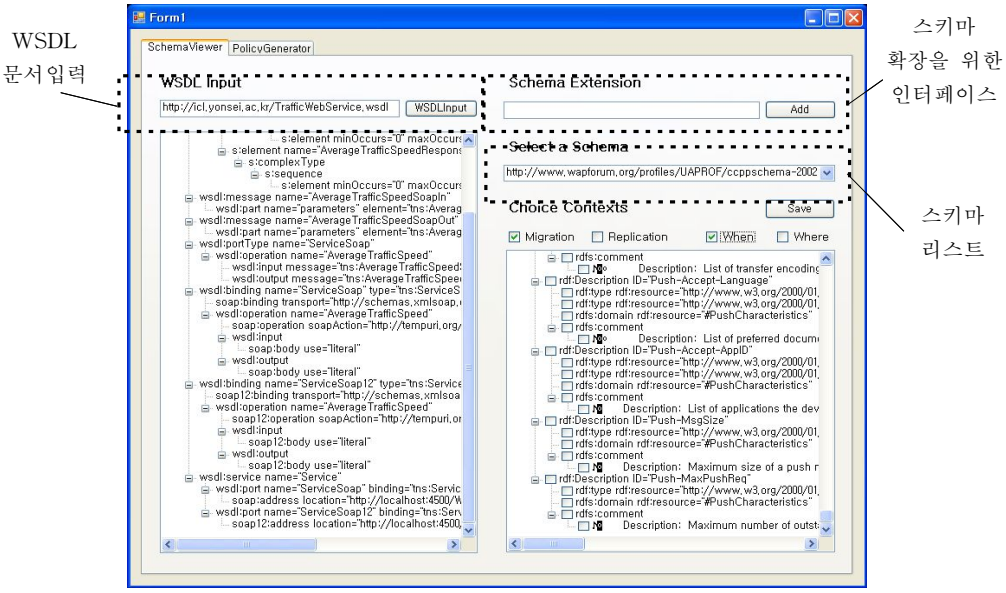
"Replication_Policy" 엘리먼트는 각각 언제, 어디로를 나타내는 "When"과 "Where" 엘리먼트를 자식 엘리먼트로 갖는다. "When"과 "Where" 엘리먼트는 이동시점과 타겟 호스트를 결정하기 위하여 필요한 하드웨어 및 소프트웨어, 그리고 네트워크 정보 등을 자식으로 갖는 엘리먼트들이다.

제안된 방법은 컨텍스트 변화에 따른 단계별 이동정책 수립을 위하여 WS-Policy의 "ExactlyOne" 및 "All" 엘리먼트를 사용한다. "ExactlyOne" 및 "All" 엘리먼트는 각각 정책집합(policy collection)과 확언집합(assertion collection)을 표현하기 위한 엘리먼트들로서 동일한 컨텍스트 일지라도 그 값에 따라 적절한 이동정책을 수립할 수 있게 한다. 또한 제안된 스키마는 UAProf을 포함한 CC/PP 프로파일 등 기존의 컨텍스트 모델을 적용하여 확장 가능하도록 설계되었다. 즉, 스키마의 확장을 통하여 서비스 제공자는 그 요구사항을 이동정책에 자유롭게 표현할 수 있다. 한편, 제안된 방법은 웹 서비스의 이동시점 및 타겟 호스트의 결정을 위하여 서비스 제공자로부터 컨텍스트의 우선순위 및 조건 연산자의 입력을 요청한다. 예를 들어, 디바이스의 성능저하로 인하여 특정 지역의 교통정보를 제공하는 웹 서비스가 이동되어야 한다고 가정할 때, 서비스 제공자는 해당 지역을 첫 번째 우선순위로, 성능을 다음 우선순위로 설정할 수 있다. 이때, 해당 컨텍스트의 "Priority" 속성을 사용한다. 또한 메모리, CPU 등과 같이 그 사용 정도에 따라 이동이 결정되어야 할 경우, 제안된 방법은 해당 컨텍스트의 "operation" 속성을 사용, 조건 연산자(Conditional operator) 즉, 크다, 작다, 크거나 같다, 작거나 같다, 같다 등을 표현함으로서 컨텍스트의 상태에 따른 이동을 결정한다.

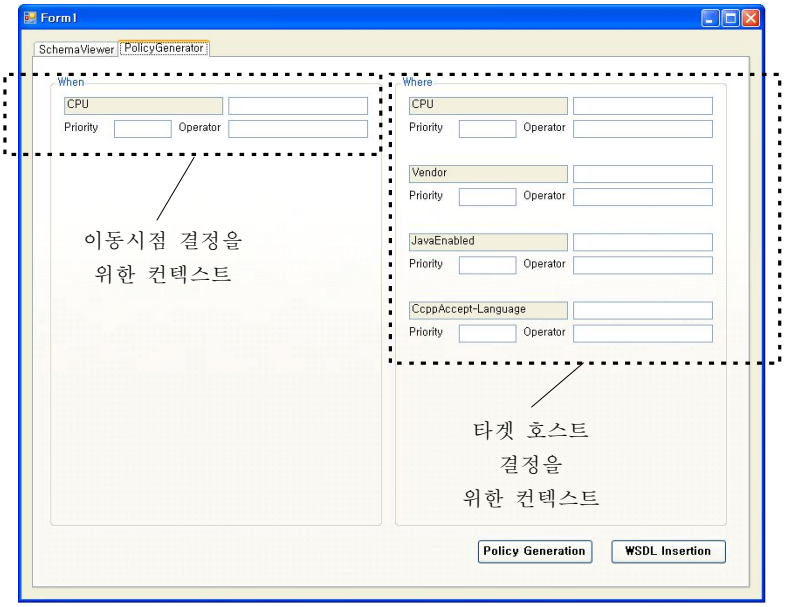


[그림 3] 웹 서비스 이동정책 설정을 위한 스키마

제안된 방법은 이동정책 수립을 위하여 [그림 4]와 같은 사용자 인터페이스를 제공한다. 사용자 인터페이스는 해당 웹 서비스의 WSDL 문서, 제안된 스키마, 그리고 서비스 제공자의 이동정책 기술을 위하여 필요한 외부 스키마를 사용하여 정책수립에 필요한 최소한의 정보를 입력받아 이동정책을 수립한 후 WSDL 문서에 이를 귀속시킨다. [그림 4(a)]는 WSDL 문서 및 외부 스키마 입력을 위한 인터페이스이며, [그림 4(b)]는 [그림 4(a)]에서 설정한 컨텍스트를 기반으로 서비스 제공자로부터 필요한 정보를 입력받기 위한 사용자 인터페이스이다. 특히, 제안된 방법은 생성된 이동정책을 해당 WSDL 문서에 적절히 삽입함으로써 웹 서비스 표준에 기반한 웹 서비스 이동이 가능하게 한다.



(a) WSDL 및 스키마 입력을 위한 인터페이스



(b) 이동정책 설정을 위한 인터페이스

[그림 4] 사용자 인터페이스

[그림 5]는 제안된 인터페이스를 통하여 생성된 이동정책의 예로서, 제안된 스키마와 WAP 포럼의 UAProf를 기반으로 한다. 생성된 이동정책은 크게 이동시점의 결정을 위한 "When" 과 타겟 호스트의 결정을 위한 "Where" 엘리먼트로 구성된다. 두 엘리먼트는 각각 서비스 제공자가 선택한 컨텍스트를 자식 엘리먼트로 갖는다. 예를 들어, [그림 5]의 이동정책은 현재 서비스를 제공하고 있는 디바이스의 CPU 사용률이 70% 이

상일 경우 이동하되, 타겟 호스트로는 자바의 사용이 가능하며, CPU 사용률이 50% 미만이고, 영어를 사용하며, HP 제품의 디바이스로 이동하라는 의미를 내포한다.

```
<mig:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
            xmlns:mig="http://icl.yonsei.ac.kr/Policy/MigrationSchema.xsd"
            xmlns:uaf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20020710#">
  <mig:Migration_Policy>
    <mig:When>
      <wsp:ExactlyOne>
        <wsp:All>
          <mig:HardwareInfo>
            <uaf:cpu operation="morethan">70%</uaf:cpu>
          </mig:HardwareInfo>
          <mig:SoftwareInfo />
          <mig:NetworkInfo />
        </wsp:All>
      </wsp:ExactlyOne>
    </mig:When>
    <mig:Where>
      <wsp:ExactlyOne>
        <wsp:All>
          <mig:HardwareInfo>
            <uaf:vendor priority="4">HP</uaf:vendor>
            <uaf:cpu priority="2" operation="lessthan">50%</uaf:cpu>
          </mig:HardwareInfo>
          <mig:SoftwareInfo>
            <uaf:CcppAccept-Language priority="3“>en</uaf:CcppAccept-Language>
            <uaf:JavaEnabled priority="1">yes</uaf:JavaEnabled>
          </mig:SoftwareInfo>
          <mig:NetworkInfo />
        </wsp:All>
      </wsp:ExactlyOne>
    </mig:Where>
  <Replication_Policy />
</mig:Policy>
```

[그림 5] 생성된 이동정책의 예

3.2 이동시점 및 타겟 호스트의 결정

특정 웹 서비스에 대한 이동정책이 수립되면 제안된 프레임웍의 이동 매니저는 수집된 컨텍스트 정보를 사용하여 이동시점을 결정한다. 즉, 이동 매니저는 생성된 이동정책으로부터 이동시점 결정을 위한 컨텍스트를 추출하고, 디바이스의 해당 컨텍스트를 모니터링 하여 이동정책을 만족하는 컨텍스트의 상태변화 시 이동을 결정하게 된다.

이동시점이 결정되면 제안된 방법은 이웃한 호스트들의 컨텍스트를 수집하고 타겟 호스트로서의 최소조건을 만족하는 후보 호스트를 추출한 후 타겟 호스트를 결정한다. 특히, 타겟 호스트의 결정은 서비스 제공자에 의해 정의된 컨텍스트의 우선순위에 기반하며, 그 값이 동일할 경우 다음 순위의 컨텍스트를 사용한다. 예를 들어, [표 1]과 같이 이동정책을 만족하며, 각각 다른 컨텍스트 값을 가진 3개의 디바이스가 있다고

가정할 때, 제안된 방법은 우선순위가 가장 높은 컨텍스트인 CPU 사용률을 검사한 후 메모리양과 대역폭을 차례로 검사한다. 이때, 제안된 방법은 현재 컨텍스트의 값과 이동정책에 정의된 값의 차가 가장 큰 호스트를 타겟 호스트로 설정한다. 따라서 3개의 디바이스 중 우선순위가 가장 높은 CPU 사용률에 의해 'A 디바이스'가 제거되고, 마지막 대역폭에 의해 그 차가 더 큰 'C 디바이스'를 타겟 호스트로 결정하게 된다.

[표 1] 타겟 호스트 설정을 위한 컨텍스트의 예

	디바이스 A	디바이스 B	디바이스 C
CPU 사용률이 50%이하, 우선순위 = 1	40%	30%	30%
사용가능한 메모리 70% 이상, 우선순위 = 2	80%	70%	70%
대역폭 100MHz 이상, 우선순위 = 3	150MHz	130MHz	140MHz

IV. 실험 결과

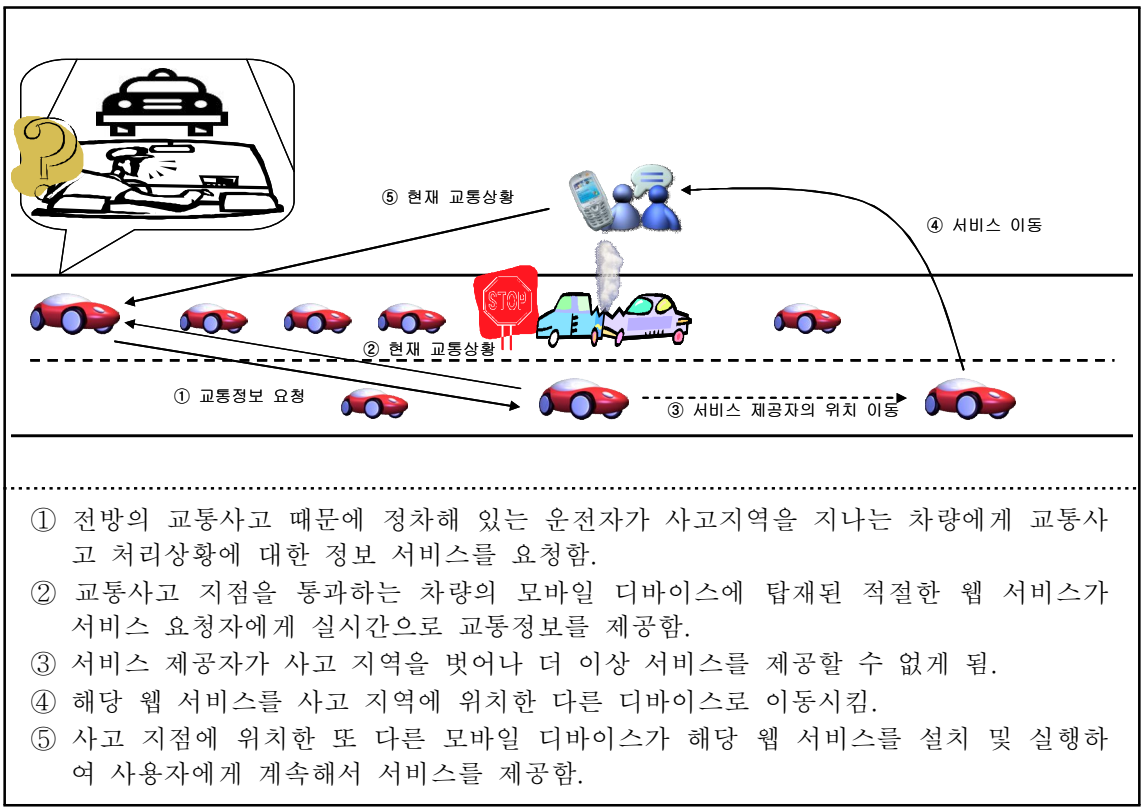
본 절에서는 디바이스간의 웹 서비스 이동을 지원하는 모바일 웹 서비스 프레임웍에 제안된 이동정책을 적용함으로서 이동정책 수립기법의 유효성을 검증한다.

4.1 실험 환경 및 시나리오

본 논문에서는 이동정책에 기반한 웹 서비스 이동의 유효성 검증을 위하여 다음과 같은 시나리오에 제안된 방법을 적용하였다. 교통사고에 의해 지체되는 고속도로에서 서비스 사용자는 사고지점의 교통상황을 알기 원한다. 서비스 사용자는 교통정보를 제공하는 웹 서비스를 찾아 사고지점으로 이동시켜 서비스를 제공받는다. 그러나 서비스를 제공하던 디바이스를 가진 자동차가 사고지점을 벗어나 더 이상 서비스를 제공할 수 없을 때 해당 웹 서비스는 자동으로 사고지점에 있는 디바이스로 이동한다([그림 6] 참조). 결과적으로 서비스 사용자는 웹 서비스의 이동과 상관없이 연속적인 서비스를 제공받는다. 제안된 방법은 [표 2]와 같이 1대의 소스 호스트와 2대의 타겟 후보 호스트로 구성된 802.11G 환경에서 이동정책의 유효성을 검증하였다.

[표 2] 실험 환경

	소스 호스트	이웃 호스트 1	이웃 호스트 2 (타겟 호스트)
제조사	HP iPAQ hx4700	Acer n50	HP iPAQ hx4700
OS	Windows Mobile 2003		
Processor	Intel PXA270 624MHz	Intel PXA272 312MHz	Intel PXA270 624MHz
Memory	64Mb RAM 128Mb ROM	64Mb RAM 64Mb ROM	64Mb RAM 128Mb ROM
위치	사고지점	사고지점으로부터 반경 50M에 위치	사고지점으로부터 반경 50M에 위치
CPU 사용률	80%	70%	30%



① 전방의 교통사고 때문에 정차해 있는 운전자가 사고지역을 지나는 차량에게 교통사고 처리상황에 대한 정보 서비스를 요청함.
② 교통사고 지점을 통과하는 차량의 모바일 디바이스에 탑재된 적절한 웹 서비스가 서비스 요청자에게 실시간으로 교통정보를 제공함.
③ 서비스 제공자가 사고 지역을 벗어나 더 이상 서비스를 제공할 수 없게 됨.
④ 해당 웹 서비스를 사고 지역에 위치한 다른 디바이스로 이동시킴.
⑤ 사고 지점에 위치한 또 다른 모바일 디바이스가 해당 웹 서비스를 설치 및 실행하여 사용자에게 계속해서 서비스를 제공함.

[그림 6] 웹 서비스 이동이 요구되는 응용 시나리오의 예

4.2 실험 결과

시나리오를 위하여 제안된 방법은 서비스 제공자로부터 웹 서비스 이동에 필요한 정보를 입력받아 [그림 7]과 같이 이동정책을 생성한 후 WSDL 문서에 귀속시킨다. 한편, 소스 호스트의 컨텍스트 매니저는 이동정책의 "When" 엘리먼트를 기반으로 적절한 컨텍스트를 추출한 후 그 변화를 모니터링하고, 이동 매니저는 이를 기반으로 이동시점을 결정한다. 이동이 결정되면 이웃한 호스트로부터 "Where" 엘리먼트에 정의된 컨텍스트를 수집하여 후보 호스트를 추출하고 다시 컨텍스트의 우선순위에 기반하여 타겟

호스트를 결정한다. [그림 7]은 교통정보 웹 서비스의 이동정책이 WSDL 문서에 귀속된 예로서, 디바이스가 사고 지점에서 100M 이상의 거리를 이동했을 때, 해당 웹 서비스는 사고 지점으로부터 반경 100M 이내에 위치하고, CPU 사용량이 50% 미만인 디바이스로 이동해야 함을 의미한다.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                                                    ....
                  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                  xmlns:mig="http://icl.yonsei.ac.kr/Policy/MigrationSchema.xsd"
                  xmlns:uaf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20020710#"
                  xmlns:dis="http://InfoMapServices.cz"
                  targetNamespace="http://tempuri.org/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <mig:Policy>
    <mig:Migration_Policy>
      <mig:When>
        <wsp:ExactlyOne>
          <wsp:All>
            <mig:HardwareInfo />
            <mig:SoftwareInfo />
            <mig:NetworkInfo />
            <dis:distance operation="morethan"> 100M </dis:distance>
          </wsp:All>
        </wsp:ExactlyOne>
      </mig:When>
      <mig:Where>
        <wsp:ExactlyOne>
          <wsp:All>
            <mig:HardwareInfo />
            <mig:SoftwareInfo />
            <mig:NetworkInfo />
            <dis:distance priority="1" operation="lessthan"> 100M </dis:distance>
            <mig:HardwareInfo>
              <uaf:cpu priority="2" operation="lessthan">50%</uaf:cpu>
            </mig:HardwareInfo>
            <mig:SoftwareInfo />
            <mig:NetworkInfo />
          </wsp:All>
        </wsp:ExactlyOne>
      </mig:Where>
    <Replication_Policy />
  </mig:Policy>

  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="AverageTrafficSpeed">
        ...
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="AverageTrafficSpeedSoapIn"> ... </wsdl:message>
  <wsdl:portType name="ServiceSoap"> ... </wsdl:portType>
  <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap"> ... </wsdl:binding>
  <wsdl:service name="Service"> ... </wsdl:service>
</wsdl:definitions>
```

[그림 7] 교통정보를 제공하는 웹 서비스의 이동정책

[그림 8]은 위 시나리오를 실제 디바이스에 적용한 결과이다. 특히, [그림 8(a)]와 [그림 8(c)]는 각각 웹 서비스가 이동한 전/후를 나타내는 소스 호스트와 타겟 호스트의 로그정보를 나타낸다. 그림에서와 같이 소스 호스트의 위치가 현재 위치에서 100M 이상 이동하면 웹 서비스의 이동이 결정된다. 이후 이동 매니저는 미리 정의된 이동정책을 기반으로 적절한 호스트 즉, 사고지점으로부터 100M 이내에 존재하고 CPU 사용률이 50% 미만인 호스트를 찾아 후보 호스트로 설정하고 이들에 대하여 다시 우선순위를 적용, 타겟 호스트를 결정한다. 실험결과 수립된 이동정책과 프레임웍에 의거 웹 서비스가 적절한 디바이스로 이동함으로서 사용자에게 지속적인 서비스를 제공함을 보였다.

(a) 소스 호스트　　(b) 후보 호스트　　(c) 후보 호스트(타겟 호스트)

[그림 8] 이동정책에 기반한 웹 서비스 이동의 예

V. 결론

무선 네트워크와 모바일 디바이스의 발달은 웹 서비스와 결합되어 모바일 웹 서비스라는 새로운 형태의 패러다임을 형성하고 있다. 그러나 불안정한 연결성 및 빈번한 컨텍스트 변화 등은 모바일 환경에서 웹 서비스를 제공하는 데 많은 문제점을 야기한다. 본 논문에서는 모바일 환경에서 웹 서비스를 효과적으로 제공하기 위하여 모바일 디바이스간의 웹 서비스 이동(migration)을 지원하는 경량의 모바일 웹 서비스 프레임웍과 함께 언제, 어디로 이동할 것인지를 결정하기 위한 웹 서비스 이동정책 수립기법을 제안하였다.

본 논문에서는 웹 서비스의 이동을 위하여 이동정책 스키마를 정의하고, 이를 기반으로 서비스 제공자로부터 최소한의 정보를 제공받아 이동정책을 생성한다. 특히, 제안된 스키마는 서비스 제공자의 요구사항을 자유롭게 표현하기 위하여 확장 가능하도록 설계되었다. 이는 스키마의 확장을 통하여 서비스 제공자의 요구사항을 이동정책에 자유롭게 표현할 수 있게 하기 위함이다. 한편, 제안된 프레임웍은 생성된 이동정책을 기반으로 컨텍스트를 모니터링 하여 웹 서비스의 이동을 자동화한다. 실험 결과, 제안된 방법으로 기술된 이동정책과 모바일 웹 서비스 프레임웍을 기반으로 웹 서비스의 이동을 통한 연속적인 서비스를 제공이 가능함을 보였다.

향후 본 연구에서는 타겟 호스트의 자동설정을 위하여 컨텍스트의 비용을 계산하는 비용모델 개발과 함께 효과적인 SOAP 통신을 위한 SOAP 메시지 최적화에 대한 연구를 진행 할 예정이다.