자율적인 웹 서비스 품질 정보 수집을 위한 프록시 클라이언트 코드의 자동 생성 방안

Automatic Generation Method of Proxy Client Code to Autonomic Quality Information Collection of Web Service

서영준*, 한정수**, 송영재***
국가기록원 대통령기록관*, 백석대학교 정보통신학부**, 경희대학교 전자정보학부***

Young-Jun Seo(yjseo@khu.ac.kr)*, Jung-Soo Han(jshan@bu.ac.kr)**,
Young-Jae Song(yjsong@khu.ac.kr)***

요약

본 논문에서는 모니터링 에이전트를 통한 웹 서비스 선정 과정의 자동화를 위해 프록시 클라이언트 코드를 자동 생성하는 방안을 제안한다. 본 논문의 기법은 템플릿 룰에 따라 WSDL 문서의 특정 엘리먼트의 속성 값을 가져옴으로써 서비스 사용자에게 프록시 클라이언트의 소스 코드를 제공할 수 있게 해 준다. 즉 XSLT 스크립트 파일은 클라이언트 코드 생성시 필요한 동적 호출 인터페이스 모델의 코드 골격을 제공한다. 이러한 코드 자동 생성 기법은 이동 에이전트 기술과 더불어 선정 아키텍처에서의 기아 상태를 해결하기 위해 필요하다. 선정 서비스를 제외하더라도 검색 결과상의 모든 서비스들에 대한 요청 HTTP 메시지를 발생시키기 위해서는 코드 자동 생성 기법이 필요하다. 생성된 프록시 클라이언트 프로그램 코드는 검색된 서비스들에 대한 더미 메시지를 발생시킨다. 본 논문에서 제시한 클라이언트 코드 생성 방안은 자동 생성 프로그래밍 영역에서의 적용 가능성을 보여준다.

■ 중심어 : | 웹 서비스 | 품질 정보 | 코드 생성 | 템플릿 룰 | WSDL |

Abstract

This paper proposes automatic generation method of proxy client code to automation of web service selection process through a monitoring agent. The technique of this paper help service consumer to provide source code of proxy client as it bring an attribute value of specific element of WSDL document using template rule. Namely, a XSLT script file provide code frame of dynamic invocation interface model. The automatic code generation technique need to solving starvation status of selection architecture. It is required to creating request HTTP message for every service on the result of search. The created proxy client program code generate dummy message about services. The proposed client code generation method show us a possibility of application in the automatic generation programming domain.

■ keyword : | Web Service | Quality Information | Code Generation | Template Rule | WSDL |

접수번호 : #071220-001
접수일자 : 2007년 12월 20일
심사완료일 : 2008년 01월 11일
교신저자 : 송영재, e-mail : yjsong@khu.ac.kr

I. 서 론

최근 웹 서비스는 기관간 연계나 기관 내부의 정보시스템 연계에서 나아가, 유비쿼터스 IT 환경을 지원하는 다양한 서비스와 분야들을 묶는 연계 기술로서 주목받고 있다. 그러나 현재 서비스 사용자들은 기능적 요구를 만족하는 웹 서비스를 찾기 위해 UDDI(Universe Description Discovery and Integration) 레지스트리를 수작업으로 검색한다. 만약 적합한 서비스가 발견된다면, QoS(Quality of Service) 정보는 UDDI 레지스트리에서 제공되지 않기 때문에 사용자는 이 정보를 얻기 위해 서비스 제공자들에 접속해야 한다. 사용자는 이러한 서비스들로부터 기능성과 QoS 요구에 가장 일치하는 하나를 수작업으로 선택할 수밖에 없다. 따라서 사용자가 실행 시간에 그들의 요구를 만족하는 서비스를 발견하도록 하기 위해서는 서비스 발견 프로세스는 자동화 되어야 한다[1]. 따라서 서비스 사용자들을 대신하여 행동하는 모니터링 에이전트와 함께 서비스 선정 작업을 자동화 하는 과정이 필요하다.

본 논문에서는 모니터링 에이전트를 통한 웹 서비스 선정 과정의 자동화를 위해 프록시 클라이언트 코드를 자동 생성하는 방안을 제안한다. 본 논문의 기법은 XSLT(eXtensible Stylesheet Language Transformations) 스크립트 파일로 표현된 템플릿 룰에 따라 WSDL(Web Services Description Language) 문서의 특정 엘리먼트의 속성 값을 가져옴으로써 서비스 사용자가 필요로 하는 구체적인 클라이언트의 소스 코드를 자동으로 생성할 수 있게 해 준다. 즉 XSLT 스크립트 파일은 클라이언트 코드 생성시 필요한 동적 호출 인터페이스 모델의 코드 골격을 제공한다. 이러한 코드 자동 생성 기법은 이동 에이전트 기술과 더불어 웹 서비스 선정 프로세스에서의 기아 상태를 해결하기 위해 필요하다. 선정 서비스를 제외하더라도 사용자 위치에서 검색 결과상의 모든 서비스들에 대한 요청 HTTP 메시지를 발생시키기 위해서는 코드 자동 생성 기법이 필요하다. 생성된 프록시 클라이언트 프로그램 코드는 이동 에이전트에 의해 실행되어 검색된 서비스들에 대한 더미 메시지를 발생시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 원격 프로시저 호출 모델과 WSDL의 구조, 소스 코드 생성패턴에 관한 이론적 배경을 소개하고, 3장에서는 에이전트의 품질 정보 수집 과정과 프록시 클라이언트 코드 생성 과정에서의 템플릿 룰에 대해 설명한다. 4장에서는 온라인 서점 웹 서비스 시스템의 신규 책 등록 예를 통해 사례 연구를 수행하였다. 마지막으로 5장에서는 결론 및 향후 연구 방향에 대해 기술한다.

II. 관련 연구

본 장에서는 WSDL의 구조와 원격 프로시저 호출 모델에 관한 기존 연구들에 대하여 소개한다.

1. WSDL의 구조

WSDL 문서의 루트 엘리먼트는 <definitions> 엘리먼트이다. 그리고 <types>, <message>, <portType>, <binding>, <service> 엘리먼트는 <definitions> 엘리먼트의 자식 엘리먼트로 구성된다. 다음 [표 1]은 [그림 1]에서 보여주는 WSDL의 문서의 구조를 이루는 핵심 엘리먼트들에 대해 간략하게 설명한 표이다[2].

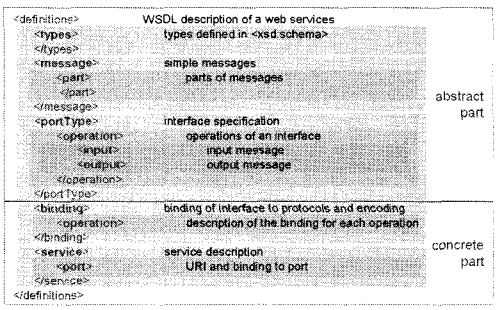


그림 1. WSDL 1.1의 구조[2]

표 1. WSDL의 핵심 엘리먼트[2]

엘리먼트	용 도
⟨definitions⟩	WSDL 문서의 루트 엘리먼트
⟨types⟩	원격 프로시저 인자 및 리턴값에서 사용될 복합 타입 기술
⟨message⟩	원격 프로시저의 인자 및 리턴값에 대한 정보 기술 인자에 대한 정보 : 인자의 수 및 인자의 데이터 타입 리턴값에 대한 정보 : 리턴값에 대한 데이터 타입
⟨operation⟩	원격 프로시저에 대한 정보 기술
⟨binding⟩	호출에 사용되는 프로토콜에 대한 정보 기술
⟨service⟩	웹 서비스 시스템의 URL(종점)을 기술

다음 [그림 2]는 WSDL 엘리먼트들 사이의 논리적 관계를 설명한 그림이다. 특히 <portType> 엘리먼트와 <binding> 엘리먼트는 참조 관계에 있다. <binding> 엘리먼트의 type 속성은 <binding> 엘리먼트에서 기술한 프로토콜을 사용해서 호출 가능한 원격 프로시저 내용을 담고 있는 <portType> 엘리먼트의 이름을 기술한다. 또한 <binding> 엘리먼트의 <operation> 엘리먼트에는 원격 프로시저 호출 및 결과 값을 표현하기 위한 네임스페이스명과 인코딩에 대한 부분을 기술해 준다. name 속성에는 <binding> 엘리먼트의 type 속성 값인 portType명에 속해 있는 원격 프로시저명을 기술해 준다.

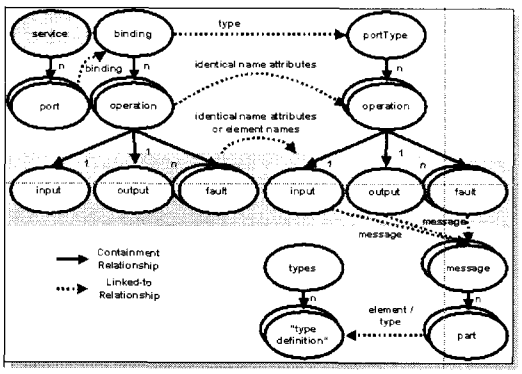


그림 2. WSDL 엘리먼트들 사이의 논리적 관계[2]

2. 원격 프로시저 호출 모델

프록시 클라이언트(Proxy Client)는 클라이언트에서 동작하는 것으로 웹 서비스 컴포넌트를 가지는 서버를 찾고, 그 서버에 존재하는 웹 서비스 컴포넌트를 검색하여 서비스의 기능을 호출하고, 매개변수와 리턴 값을 전달하는 기능을 가진다. CORBA의 Skeleton/Stub 코드 개념 중 Stub 개념에 해당한다. 웹 서비스 시스템을 이용하는 프록시 클라이언트를 JAX-RPC로 개발할 경우에는 다음 세 가지 원격 프로시저 호출 모델 중 하나를 선택할 수 있다[3].

- 정적 스텁 호출 모델 : 정적 스텁이란 WSDL 문서를 이용해서 수동으로 생성된 스텁 클래스를 말한다. 정적 스텁 호출 모델은 생성된 스텁을 이용해서 원격 프로시저 호출을 하는 모델이다.
- 동적 프록시 호출 모델 : 동적 프록시란 동적으로 생성되는 스텁이라 할 수 있다. WSDL 문서가 필요하다는 점에서 정적 호출 모델과 비슷하나 수동으로 스텁 클래스를 생성하지 않고 실행 도중에 스텁 객체가 생성된다.
- 동적 호출 인터페이스 모델 : 클라이언트 측에서 원격 인터페이스명을 알지 못해도 동적으로 원격 메소드를 호출할 수 있도록 해주는 모델이다. 다음 [그림 3]은 동적 호출 인터페이스 모델을 통한 서비스 클라이언트 개발 과정을 그린 그림이다.

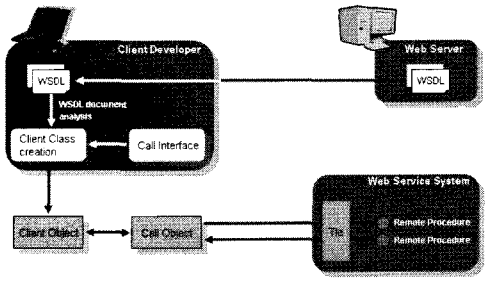


그림 3. 동적 호출 인터페이스의 이해[2]

3. 소스 코드 생성 패턴

[4]의 연구에서는 공통적으로 사용하는 소스 코드 생성 기법을 기술하는 패턴들의 집합을 소개한다. Templates+Filtering 패턴은 코드를 생성하기 위한 가장 간단한 방법을 기술하며, 코드는 명세의 일부 부분을 필터링 한 후에 텍스트 모델 명세(XML/XMI)에 템플릿을 적용함으로서 생성된다. Templates+MetaModel 패턴은 Templates+Filtering 패턴의 확장 형태이다. 직접 모델에 패턴을 적용하는 대신에, 명세로부터 메타모델을 실체화 하며, 템플릿은 메타모델에 의하여 기술된다. Frame Processing 패턴은 프레임에 의하여 코드를 생성하며, 프레임은 다른 프레임 인스턴스 뿐만 아니라 숫자와 문자열에 의해 매개변수화 될 수 있다. API-based generateors 패턴은 코드생성 프로그램이 작성되는 것에 대비하여 API를 제공한다. API는 전형적으로 타겟 언어의 메타모델/구문에 기반한다. 마지막으로 Inline Code generation 패턴은 코드 생성이 전처

리기에 의하여 컴파일 동안 행해진다. 순차적으로 컴파일 되거나 인터프리트 되는 프로그램을 수정한다.

[5]의 연구에서는 설계 컴포넌트의 재사용성을 최대화하기 위하여 디자인 패턴을 대상으로 호환성을 가지는 코드를 생성하는 툴을 제안하였다. 제안된 툴은 패턴의 설명 정보와 함께 추상형의 구조 정보를 저장하는 라이브러리를 구축한다. 패턴의 구조 정보는 특정 애플리케이션에 적합하게 실체화 하는 과정을 거친다. 실체화된 구조 정보는 코드 생성 템플릿을 통해 XMI 코드로 생성된다. XMI는 대부분의 CASE 툴에서 변환 포맷으로 지원되므로, 호환성을 보장 받을 수 있다.

III. 프록시 클라이언트 코드의 자동 생성 방안

1. 에이전트의 품질 정보 수집

서비스 사용자와 제공자 사이의 SOAP 메시지는 전송 프로토콜에 의해 배달이 되어야 한다. SOAP과 전송 프로토콜을 접목 시키는 작업을 바인딩(binding)이라고 한다. 이론적으로 SOAP 메시지는 어떠한 전송 프로토콜과도 바인딩 될 수 있으나, HTTP 전송 프로토콜과의 바인딩이 일반적이다. HTTP 프로토콜과의 바인딩 후 HTTP의 내용 중에는 SOAP 메시지의 요청과 응답 과정에 대한 부가 정보가 들어 있다. 에이전트는 이러한 부가 정보를 수집해 전달함으로서 브로커 서버가 해당 웹 서비스의 품질 척도를 측정할 수 있도록 지원한다. 즉 에이전트가 웹 서비스의 품질 정보를 측정하기 위해서는 서비스 사용자가 제공자에게 보내는 SOAP 메시지의 목적지 주소를 서비스 제공자가 아닌 에이전트가 수신하는 주소로 변경되어야 한다. 따라서 SOAP 메시지를 발생하기 위해 서비스 소비자에서 실행하는 프록시 클라이언트의 소스 코드를 자동 생성함으로서 에이전트에게 SOAP 메시지 전달이 가능하다.

모니터링 에이전트는 [그림 4]와 같이 서비스 사용자에서 보내는 요청 HTTP 내용을 받아서 바인딩 정보를 수집 하고, 똑 같은 내용을 서비스 제공자 쪽으로 다시 보낸다. 마찬가지로 서비스 제공자 쪽에서 오는 응답 HTTP 내용에서 바인딩 정보를 수집하고 똑같은 내용을 서비스 사용자에게로 다시 보내 준다.

요청 SOAP 메시지는 요청 HTTP에 실어 보내고, 응답 SOAP 메시지는 응답 HTTP에 담겨서 전송된다. HTTP는 시작 행, 헤더, 본문으로 크게 3 부분으로 나뉘며, 요청 HTTP와 응답 HTTP의 구조는 동일하다 [HTTP97]. 단, 시작 행을 이루고 있는 구성 요소는 요청 시와 응답 시 다르다. 특히, 응답 시의 시작 행(Status-Line)의 구성 요소 중 상태 코드(Status Code)에는 클라이언트의 요청에 대한 성공 여부를 나타내는 코드가 온다.

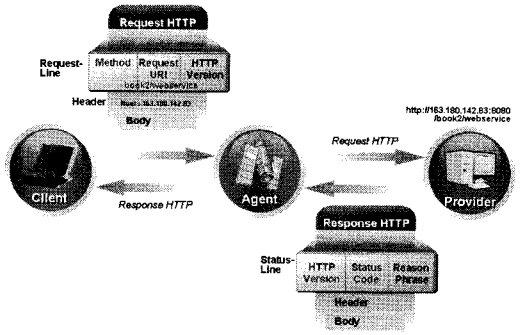


그림 4. 모니터링 에이전트의 패킷 정보 수집

2. 변환 처리 과정

본 논문에서는 일반적으로 사용되는 소스 코드 생성 기법 중 Templates+Filtering 패턴을 사용한다[3]. Templates+Filtering 패턴은 필터링 메커니즘이 강력하고 명세가 잘 되어 있다면 효율적인 방법이며, XML과 XSLT와 같은 산업 표준 툴 상에 기반하고 있다는 장점이 있다. 그러나 템플릿으로 사용되는 XSLT 구문은 읽기 어렵고 많은 질의가 수행된다면 너무 복잡해진다. 또한 코드 생성 템플릿이 명세 구문의 저 레벨 상세함에 의존하는 단점이 있다[4].

본 논문에서 프록시 클라이언트 소스 코드는 웹 서비스 명세의 일부 부분을 필터링 한 후에 템플릿을 적용함으로서 생성된다. 템플릿의 목적은 프록시 클라이언트 프로그램에서 동적 호출 인터페이스 모델을 자동 생성하기 위함이다. 따라서 템플릿을 설계하기 이전에 동적 호출 인터페이스 부분이 해야 하는 일을 명확히 제

시하고 그에 따라 템플릿을 제안해야 한다.

다음 [그림 5]는 동적 호출 인터페이스 모델의 프록시 클라이언트 코드 생성 과정을 도식화한 것이다. 변환 과정의 중추적 역할을 하는 변환기는 최근 활발히 연구되는 소프트웨어 자동 생성 프로그래밍 분야에 적용되기 시작한 XSL 변환기를 사용하였다[6][7]. 템플릿은 사용자가 작성하고자 하는 프록시 클라이언트 프로그램의 동적 호출 인터페이스 모델의 정보를 나타내며, 이 정보를 입력받은 XSL 변환기는 선정된 서비스를 포함한 검색 결과상의 서비스들의 웹 서비스 명세(WSDL)를 바탕으로 동적 호출 인터페이스 모델의 프록시 코드를 자동 생성하는 기능을 수행한다. 즉, 웹 서비스 명세는 XSL 변환기의 입력으로 사용되어 주어진 템플릿에 근거하여 프록시 클라이언트 프로그램의 동적 호출 인터페이스 모델의 골격 코드를 생성 한다. 선정되지 않은 서비스들의 프록시 클라이언트 프로그램들은 각 서비스들의 성능 정보를 알기 위한 더미 메시지(dummy message)를 발생시키기 위해 사용된다.

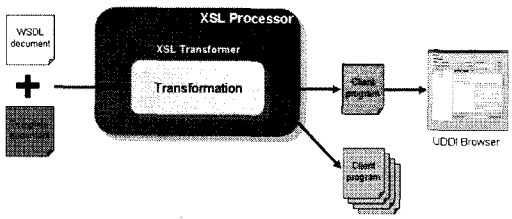


그림 5. 변환 처리 과정

3. 시작 템플릿 룰

본 논문에서 템플릿은 시작 템플릿 룰과 원격 프로시저를 기술한 soapbinding 템플릿 룰로 구성된다. [그림 6]은 시작 템플릿 룰을 정의하는 그림이며, soapbinding 템플릿 룰을 적용하기 위해서는 해당 부분에 xsl:call-template라는 엘리먼트를 작성해 주어야 한다. XSL 변환기는 우선 name 속성에 지정되어 있는 대상 노드와 동일한 노드를 name 속성으로 가지고 있는 템플릿 룰을 찾으며, 그 템플릿 룰을 적용해서 변환된 내용을 xsl:call-template 엘리먼트와 대치시킨다.

```
01:  <?xml version="1.0" ?>
02:  - <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
03:      xmlns:msxsl="urn:schemas-microsoft-com:xslt" xmlns:fo="urn:p2plusfo:xslfo#formats"
04:      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://www.w3.org/2001/XMLSchema"
05:      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soap-
06:      enc="http://schemas.xmlsoap.org/soap/encoding/"
07:      xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
08:    <xsl:strip-space elements="*" />
09:    <xsl:output method="text" version="4.0" />
10:  - <xsl:template match="/">
11:      import javax.xml.rpc.*; import javax.xml.namespace.*; public class BookClient { public static void
12:      main(String args[]) throws Exception { ServiceFactory serviceFactory =
13:      ServiceFactory.newInstance(); QName serviceQName = new QName("
14:      <xsl:value-of select="/wsdl:definitions/@targetNamespace" />
15:      ",
16:      <xsl:value-of select="wsdl:definitions/wsdl:service/@name" />
17:      "); Service service = serviceFactory.createService(serviceQName); QName portQName = new
18:      QName("
19:      <xsl:value-of select="/wsdl:definitions/@targetNamespace" />
20:      ",
21:      <xsl:value-of select="wsdl:definitions/wsdl:service/wsdl:port/@name" />
22:      "); Call call = service.createCall(portQName); call.setTargetEndpointAddress(
23:      <xsl:value-of select="concat('http://localhost:9090',substring
24:        (wsdl:definitions/wsdl:service/wsdl:port/soap:address/@location, 27))" />
25:      ); call.setProperty(Call.SOAPACTION_USE_PROPERTY, new Boolean(true)); call.setProperty
26:      (Call.SOAPACTION_URI_PROPERTY, ""); call.setProperty
27:      ( "javax.xml.rpc.encodingstyle.namespace.uri", "http://schemas.xmlsoap.org/soap/encoding/");
28:      <xsl:for-each select="/wsdl:definitions/wsdl:binding[@name = substring-after
29:        (/wsdl:definitions/wsdl:service/wsdl:port/@binding, ':')]">
30:        <xsl:call-template name="soapbinding" />
31:      </xsl:for-each>
32:      ) }
33:    </xsl:template>
34:  + <xsl:template name="soapbinding">
35:    </xsl:stylesheet>
```

그림 6. 시작 템플릿 룰 정의

다음 [그림 7]은 템플릿 룰에 대한 수행 과정 설명이다.

09: 결과 트리의 문서 종류를 텍스트 문서로 지정
10-33: 문서 전체를 변환시키는 템플릿 룰 정의
12: 웹 서비스 시스템에 대한 정보를 담고 있는 Service 객체를 생성하는 ServiceFactory 객체 생성
13-16: 웹 서비스 시스템명을 QName 객체로 만듦. WSDL 문서에서 웹 서비스 네임스페이스명과 웹 서비스명을 가져옴
17: Service 객체 생성
18-21: 웹 서비스 포트명을 QName 객체로 만듦 WSDL 문서에서 웹 서비스 네임스페이스명과 포트명을 가져옴
22: Call 객체 생성
23-24: 웹 서비스 시스템의 URL(종점:endpoint) 지정. WSDL 문서에는 종점의 포트 번호가 8080번이므로, 에이전트의 listenPort인 9090번으로 변경
25: SOAPACTION 사용 여부 지정
26: SOAPACTION URI 지정
27: SOAP 메시지 인코딩 지정
28-31: Call 객체를 통해 원격 메소드를 호출하는 soapbinding 템플릿 룰을 적용. WSDL 문서의 <binding> 엘리먼트의 바인딩 이름을 기술한 name 속성 값과 <service>/<port> 엘리먼트의 binding 속성 값이 일치하는 노드를 대상으로 한다. 대상 노드들의 숫자만큼 반복해서 템플릿 룰 적용

그림 7. 시작 템플릿 룰 수행 프로세스

[그림 7]의 23-24 라인에서 설명하는 listenPort의 변경은 이동 에이전트의 모니터링 활동을 위해서는 필수적인 과정이다. 에이전트가 수행 중인 로컬 호스트의

특정 listenPort로의 포트 변경으로 클라이언트 애플리케이션과 웹 서비스 시스템 사이에서 전달되는 SOAP 메시지는 이동 에이전트를 통해 이루어진다.

4. soapbinding 템플릿 룰

다음 [그림 8]에서 정의한 soapbinding 템플릿 룰에서는 Call 객체의 속성 값을 지정하는 마지막 과정으로 원격 프로시저 관련 속성 값을 세팅해 준다. WSDL 문서에 기술된 원격 프로시저마다 원격 프로시저명을 지정하고, 각 프로시저의 인자명, 인자형, 그리고 반환형을 세팅한다.

```
01: - <xsl:template name="soapbinding">
02:     <xsl:variable name="portTypeName" select="substring-after(current()/@type, ':')" />
03:   - <xsl:for-each select="wsdl:operation">
04:       <xsl:variable name="inputMessageName" select="substring-after(/wsdl:definitions/wsdl:portType
05:         [@name = $portTypeName]/wsdl:operation[@name = current
06:         ()/@name)/wsdl:input/@message, ':')" />
07:       <xsl:variable name="outputMessageName" select="substring-after(/wsdl:definitions/wsdl:portType
08:         [@name = $portTypeName]/wsdl:operation[@name = current
09:         ()/@name)/wsdl:output/@message, ':')" />
10:     - <xsl:for-each select="/wsdl:definitions/wsdl:portType[@name = $portTypeName]/wsdl:operation
11:         [@name = current()/@name]">
12:         call.setOperationName(new QName("
13:         <xsl:value-of select="/wsdl:definitions/@targetNamespace" />
14:         ", "
15:         <xsl:value-of select="current()/@name" />
16:         "));
17:       - <xsl:for-each select="/wsdl:definitions/wsdl:message[@name = $inputMessageName]">
18:           call.addParameter("
19:           <xsl:value-of select="current()/wsdl:part/@name" />
20:           ", new QName("http://www.w3.org/2001/XMLSchema", "
21:           <xsl:value-of select="substring-after(current()/wsdl:part/@type, ':')" />
22:           "),ParameterMode.IN);
23:         </xsl:for-each>
24:       - <xsl:for-each select="/wsdl:definitions/wsdl:message[@name = $outputMessageName]">
25:           call.setReturnType(new QName("http://www.w3.org/2001/XMLSchema", "
26:           <xsl:value-of select="substring-after(current()/wsdl:part/@type, ':')" />
27:           "));
28:         </xsl:for-each>
29:       </xsl:for-each>
30:     </xsl:for-each>
31: </xsl:template>
```

그림 8. soapbinding 템플릿 룰 정의

[그림 9]는 soapbinding 템플릿 룰에 대한 수행 과정 설명이다. [그림 9]의 10-29 라인의 반복문에서는 조건에 맞는 원격 프로시저명을 찾아 관련 속성 값들을 세팅하는 코드이다. 조건은 <portType> 엘리먼트의 name 속성은 portType의 이름을 기술하는데, 이 이름은 <binding> 엘리먼트에서 참조되므로 같아야 한다. 또한 <binding> 엘리먼트의 <operation> 엘리먼트의 name 속성에는 <binding> 엘리먼트의 type 속성 값인 portType명에 속해 있는 원격 프로시저명을 기술해 주므로 두 속성 값이 일치하는 경우에 대상 노드를 찾는다.

01-31: 원격 프로시저를 기술하는 템플릿 룰 정의
02: portTypeName 변수 선언. 현재 노드(<binding> 엘리먼트)의 type 속성에는 <binding> 엘리먼트에서 기술한 프로토콜을 사용해서 호출 가능한 원격 프로시저 내용을 담고 있는 <portType> 엘리먼트의 이름을 기술
03-30: <operation> 엘리먼트들의 숫자만큼 반복해서 수행.
04-06: inputMessageName 변수 선언. <portType> 엘리먼트의 name 속성이 portTypeName 변수 값과 일치하는 경우 <operation> 엘리먼트의 인자 타입을 기술한 message 엘리먼트 참조
07-09: outputMessageName 변수 선언. <portType> 엘리먼트의 name 속성이 portTypeName 변수 값과 일치하는 경우 <operation> 엘리먼트의 리턴 타입을 기술한 message 엘리먼트 참조
10-29: <portType> 엘리먼트의 name 속성이 portTypeName 변수 값과 일치하는 노드들의 숫자만큼 반복. 원격 프로시저 네임스페이스명과 원격 프로시저명을 인자로 하여 원격 프로시저 이름 지정
17-23: <message> 엘리먼트의 name 속성 값과 inputMessageName 변수 값이 일치하는 노드들의 숫자만큼 반복. 원격 프로시저 인자형을 지정. 인자명은 <part> 엘리먼트의 name 속성 값으로, 인자형은 type 속성 값으로 기술
24-28: <message> 엘리먼트의 name 속성 값과 outputMessageName 변수 값이 일치하는 노드들의 숫자만큼 반복. 원격 프로시저 반환형을 지정. 반환형은 type 속성 값으로 기술

그림 9. soapbinding 템플릿 룰 수행 프로세스

Ⅳ. 사례 연구

다음에서 설명하는 온라인 서점 웹 서비스 시스템을 예로 들어 변환 처리 과정을 설명하고자 한다. 온라인 서점 웹 서비스 시스템은 신규 책에 홍보하기를 원하는 출판사가 언제든지 온라인 서점의 웹 서비스 시스템에 바인딩 하여 원격 프로시저를 호출함으로써 자동으로 신규 책을 등록 할 수 있도록 만들어 주는 시스템이다. 다음 [그림 10]은 BookIF를 구현한 BookImpl 구현 클래스의 구조이다. addBook1은 한 개의 신규 책에 대한 제목과 가격을 인자로 받아서 저장하는 원격 프로시저이다. addBook2는 여러 개의 신규 책에 대한 제목을 배열 인자로 받아서 한꺼번에 저장하는 원격 프로시저이다. addBook3는 구조체인 Book.class형의 인자를 받아

서 저장하는 원격 프로시저이다.

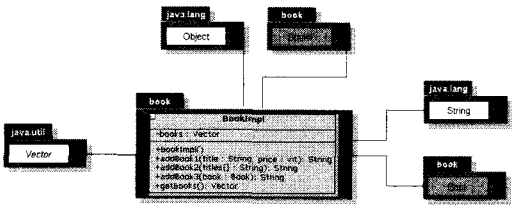


그림 10. BookImpl 클래스의 구조

다음 [그림 11]은 템플릿 문서와 WSDL 문서간의 매핑을 설명한 그림이다. 템플릿 문서에 정의된 템플릿 규칙의 지시 엘리먼트에서는 WSDL 문서의 대상 엘리먼트의 속성 값을 가져와 대치시키게 된다. 따라서 [그림 11]에서는 [그림 10]에서 설명한 온라인 서점 웹 서비스 시스템을 예로 들어 대치되는 속성 값들을 원문자 번호로 표기하였다. [그림 11]의 가운데 있는 사각형은 [그림 9]에서 설명한 10-29 라인의 반복문의 조건식을 도식화 한 것이다. <portType> 엘리먼트와 <binding> 엘리먼트 사이에서 두 가지 속성 값(portType의 이름, operation 이름)들을 비교하여 해당되는 원격 프로시저명을 대치한다.

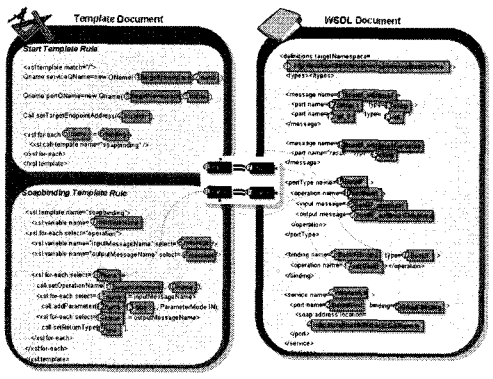


그림 11. 템플릿과 WSDL 문서간 매핑

다음 [그림 12]는 자동 생성된 BookClient.java의 전체 소스 코드이다. 단 Call 객체를 통해 인터페이스에 정의된 원격 메소드를 호출하는 마지막 구문은 서비스 사용자에 의해 추가되어야 한다.

```
import javax.xml.rpc.*;
import javax.xml.namespace.*;

public class BookClient {
    public static void main(String args[]) throws Exception {
        ServiceFactory serviceFactory = ServiceFactory.newInstance();
        QName serviceQName = new QName(
            "http://163.180.142.83:8080/book2/webservice/wsdl/webservice",
            "Webservice");
        Service service = serviceFactory.createService(serviceQName);
        QName portQName = new QName(
            "http://163.180.142.83:8080/book2/webservice/wsdl/webservice",
            "BookIFPort");
        Call call = service.createCall(portQName);

        call.setTargetEndpointAddress(
            "http://localhost:9092/book2/webservice");
        call.setProperty(Call.SOAPACTION_USE_PROPERTY,
            new Boolean(true));
        call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
        call.setProperty("javax.xml.rpc.encodingstyle.namespace.uri",
            "http://schemas.xmlsoap.org/soap/encoding/");
        call.setOperationName(
            new QName(
                "http://163.180.143.89:8080/book2/webservice/wsdl/webservice",
                "addBook1");
        call.addParameter("String_1", new QName(
                "http://www.w3.org/2001/XMLSchema", "string"),
                ParameterMode.IN);
        call.addParameter("int_2", new QName(
                "http://www.w3.org/2001/XMLSchema", "int"),
                ParameterMode.IN);
        call.setReturnType(new QName(
            "http://www.w3.org/2001/XMLSchema", "string"));

        Object params[] = { new String("EJB Programming"),
            new Integer(29000) };

        String result = (String) call.invoke(params);
        System.out.println(result);
        }
}
```

그림 12. BookClient.java

V. 결론

본 논문에서는 웹 서비스 명세 문서인 WSDL 문서와 XSLT 기술을 이용하여 클라이언트 코드 생성 시에 동적 호출 인터페이스 모델을 지원하는 코드 자동 생성 기법을 제안하였다. 이러한 코드 자동 생성 기법은 모니터링 에이전트의 자율적인 성능 정보 수집에 필요한 기법이다. 본 논문에서는 Templates+Filtering 패턴을 사용하여 작성된 템플릿을 통해 선정된 서비스를 포함한 검색 결과상의 서비스들의 웹 서비스 명세(WSDL)를 바탕으로 동적 호출 인터페이스 모델의 프록시 코드를 자동 생성한다.

향후 연구로는 생성된 코드를 통해 서비스 사용자들로부터 품질 정보를 수집하기 위한 이동 에이전트의 설계, 구현이 필요하다.

참 고 문 헌

[1] E. M. Maximilien and M. P. Singh, "A Framework and Ontology for Dynamic Web Services Selection," IEEE Internet Computing, Vol.8, No.5, pp.84-93, 2004.

[2] http://www.w3.org/2002/ws/desc/

[3] 신민철, *기초에서 실무까지 XML 웹 서비스*, (주)프리렉, 2004.

[4] M. Voelter, "A Catalog of Patterns for Program Generation", Proceedings of the EuroPLoP conference, 2003.

[5] Y.J. Seo and Y.J. Song, "A Study on Automatic Code Generation Tool from Design Pattern Based on the XMI," LNCS Vol.3983, pp.864-872, Springer-Verlag, 2006.

[6] J. C. Cleaveland, "Program Generators with XML and Java," Prentice Hall, 2001.

[7] http://www.w3.org/TR/xslt

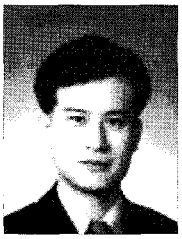
저 자 소 개

서 영 준(Young-Jun Seo) 정회원

- 1999년 2월 : 경희대학교 전자계산공학과(공학사)
- 2001년 2월 : 경희대학교 대학원 전자계산공학과(공학석사)
- 2007년 8월 : 경희대학교 대학원 전자계산공학과(공학박사)
- 2007년 11월 ~ 현재 : 국가기록원 대통령기록관 공업연구사

<관심분야> : 웹 서비스, CBSE, 소프트웨어 재사용

한 정 수(Jung-Soo Han) 종신회원



- 1990년 : 경희대학교 전자계산공학과(공학사)
- 1992년 : 경희대학교 대학원 전자계산공학과(공학석사)
- 2000년 : 경희대학교 대학원 전자계산공학과(공학박사)
- 2001년 ~ 현재 : 백석대학교 정보통신학부 교수

<관심분야> : 웹 서비스, CBSE, CASE 도구

송 영 재(Young-Jae Song) 정회원

- 1969년 : 인하대학교 전기공학과(공학사)
- 1976년 : 일본 Keio University 대학원 전산학과(공학석사)
- 1979년 : 명지대학교 대학원 전산학과(공학박사)
- 1971년 ~ 1973년 : 일본 Toyo Seiko 연구원
- 1982년 ~ 1983년 : 미국 Maryland University 전산학과 연구교수
- 1984년 ~ 1989년 : 경희대학교 전자계산소장
- 1989년 ~ 1990년 : 일본 Keio University 전산학과 객원교수
- 1993년 ~ 1995년 : 경희대학교 교무처장
- 1996년 ~ 1998년 : 경희대학교 공과대학장
- 1998년 ~ 2000년 : 경희대학교 기획조정실장
- 2001년 ~ 2002년 : 경희대학교 산업정보대학원장
- 1976년 ~ 현재 : 경희대학교 전자정보대학 교수

<관심분야> : 웹 서비스, CBSE, CASE 도구, 소프트웨어 재사용