상황정보 적응을 위하여 다형성을 이용한 SOA 웹서비스 구현 방안☆

A SOA Web Service Implementation for Adapting to Context Changes using Polymorphism

윤 회 진*
Hoijin Yoon

요 약

본 논문은 웹서비스로 구현되는 상황인식 어플리케이션에서 상황정보에 적응하는 방안으로서 다형성을 적용하여, 웹서비스 통합을 구현하고, 그를 통한 비용 감소의 효과를 보인다. 유비쿼터스 컴퓨팅의 대두로, 상황인식어플리케이션에 대한 관심이 높아지고 있으며, 그에 따른 상황정보 적응성 보장이 주요한 이슈로 등장하고 있다. 상황정보가 꾸준히 변화되어지고, 어플리케이션은 그 변화에 적응해야 하는데, 이때 보다 적은 비용으로 적응성을 해결하려고 하는 노력을 요구된다. 이는 최근 주목을 받는 서비스 기반 아키텍처를 통하여 웹서비스 기술을 보다 체계적으로 접근할 수 있다. 웹서비스로 구현되는 상황인식 어플리케이션의 경우, 서비스들의 통합 구현에 상황인식 적응성을 반드시 고려해야 하며, 나아가 보다 적은 비용으로 해결할 수 있는 방안을 요구하게 된다. 본 논문은 객체지향의 다형성 기술을 통하여, 통합 서비스를 구현하는 방안을 제안하고, 구체적인 웹서비스 구현을 분석하여, 제시한 통합 서비스 구현 방안이 적응성을 구현에 어느 정도 수행 코드 감소를 가져오는지를 보인다.

Abstract

This paper proposes an implementation of an orchestration service of web services, which work as context aware applications in Ubiquitous Computing. The proposed method aims to decrease the cost of adapting to the context changes. UC requires the context adaptation more than any other computing environments, and it works based on networks of various heterogeneous platforms. Web services on SOA could be one of the solutions for the requirements. This paper proposes the way how to implement an orchestration service of SOA to handle the context adaptation while keeping the performance high. We develops an implementation model of orchestration services of web services, and the model decreases the number of code lines of the orchestration service. This paper also shows a simple empirical study, and it also analyzes two different method implementing the polymorphism, overloading and overriding.

☞ keyword : 상황정보적응성, 웹서비스, 다형성, 서비스기반아키텍처

1. 서론

HP의 쿨타운 [1]의 경우처럼 웹환경에서 수행되는 상황인식의 경우 웹서비스 기술의 이용이 필수적이다. 웹서비스 기술을 이용하여 느슨한 결합과 추상화 등의 특성을 지원하는 서비스 기반 아키텍처 (Service-Oriented Architecture : SOA)가 최근 이슈가 되고 있으며, SOA는 가트너그룹이 예측하는 향후 3년이내에 최고 정점에 도달할 유망한 기술이다 [2]. 상황인식 어플리케이션은 다양한 플랫폼들 사이의 상호운용성을 기본으로 한다. 따라서 플랫폼 독립적인 XML기반 메세지 전송을 통한 서비스 호출 및 수행 구조인 SOA 웹서비스가 상황인식 어플리케이션 구현에 적합하다 [3].

상황인식 어플리케이션은 상황정보의 변화를 실시간으로 인식하여 적합한 행위를 해야 한다[4].

* 정 회 원 : 협성대학교 컴퓨터공학과 교수(교신저자)
hjyoon@uhs.ac.kr
☆ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2007-331-D00406)
[2008/01/28 투고 - 2008/02/26 심사 - 2008/05/15 심사완료]

기존의 분산 컴포넌트 모델들은 소프트웨어 컴포넌트들 사이의 인터페이스를 명세하고 각 컴포넌트들 사이의 주고받는 인터페이스를 맞추는 방식으로 컴포넌트들을 조립하였다. 이는 실행시간 전에 이미 전체 시스템이 고정적으로 결정되는 단점을 갖는다. 그러나 SOA는 서비스들은 각각이 독립적으로 수행될 수 있는 자율성(autonomous)의 특성을 가지므로 다른 서비스와의 연동을 전혀 고려하지 않고, 서비스들 사이의 느슨한 결합(loosely-coupled) 관계를 가진다. SOA는 통합 서비스(orchestration service)에서 필요한 서비스들에게 SOAP 메세지를 전송하는 방식으로 서비스를 수행시키므로 실행시간 중에 메세지 전송의 대상을 변경함으로써, 수행되는 서비스를 변경할 수 있다. 그러므로 UC에서 상황정보에 대한 서비스를 실시간으로 찾아 메세지 전송을 하도록 하는 상황정보의 적응성 구현에 SOA가 적합하다.

상황정보의 적응성을 서비스 내에 어떻게 구현할지를 고려해야 한다[5]. Sommerville이 예로 든 in-car information 시스템의 경우와 같이, 상황인식 어플리케이션은 장소의 이동, 시간의 변화, 사용자 선호도 변화 등의 상황정보의 실시간 변화를 인식하고, 현재의 상황정보에 적합한 구현을 갖는 서비스를 동적으로 찾아 바인딩하여야 한다. 따라서 적합한 서비스를 찾고, 그 서비스를 요청하고 답을 받는 일련의 과정들이 어플리케이션을 이루는 서비스 내에 모두 구현되어져야 한다. 또한, 서비스 수행 성능을 고려해야 한다. 상황인식 어플리케이션은 실시간 변화에 민감하게 반응해야 하므로 수행 성능이 중요한 반면, SOA 구현 기술인 웹서비스의 SOAP 메세지 전송 오버헤드가 수행 성능을 저해한다[6].

상황정보의 적응성을 구현하기 위하여 SOA 통합 서비스가 웹서비스 검색부터 SOAP 메세지 전송까지 수행하는 코드를 포함하여야 하며, 이를 위해 다양한 API들을 사용해야만 하고 웹서비스가 바뀔 때 마다 WSDL분석과 SOAP 메세지 작성을 반복수행해야 하므로, 이는 수행 성능을 오히려 저해하는 요인이 된다. 상황정보의 적응성과 성능을 모두 고려한 방안으로, 본 논문은 "one interface multiple implementations"를 구현하는 다형성을 웹서비스 시스템에 적용하고, 이를 통해 통합 서비스에서 수행하는 상황정보의 적응성 절차를 줄이는 효과를 얻는다.

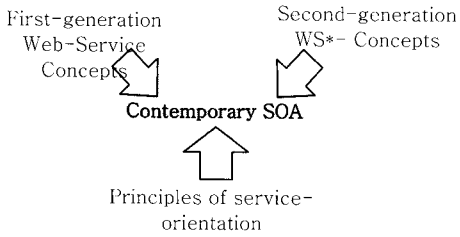
2장에서는 상황인식에 대한 적응성 관련 연구들과 웹서비스의 성능에 관한 기존의 연구들을 살펴보고 이들 연구와 본 연구와의 차이를 언급한다. 3장은 본 논문에서 제안하는 웹서비스의 다형성 구현 모델을 구체적으로 설명하고, 4장은 우리의 방법을 적용하는 경우와 아닌 경우에 나타나는 수행 절차 수를 비교 분석하여 다형성 적용이 갖는 통합 서비스의 수행 성능 향상 효과를 설명한다. 마지막 5장의 결론으로 본 논문이 제안한 방법이 갖는 의미와 제약점을 정리한다.

2. 관련연구

2.1 SOA 웹서비스 성능향상 연구

SOA는 1992년에 이미 제안되었지만, 그 당시의 구현 기술의 부재로 개발현장의 관심을 받지 못했으나, 그 후 웹서비스 기술을 통한 구현 가능성에 힘입어 최근 각광을 받고 있는 신개념의 소프트웨어 아키텍쳐이다. 그림 1은 현재 언급하는 SOA를 설명하고 있다. SOA는 기존의 구현기술로서의 웹서비스와 WS-Coordination등의 웹서비스 QOS (Quality of Service) 권고안을 나타내는 WS*(Web Service Star), 그리고 서비스 지향이 갖는 기본 원칙들 모두를 수용해야 한다[7]. 서비스 지향이 갖는 기본 원칙에는 자율성, 약한 결합력, 민첩성, 상호운용성 등이 포함된다. 본 논문도 기존의 publish-subscribe 모델로서의 웹서비스 기술이 아닌 WS-Orchestration과 서비스 지향의 기본 개념을 고려하여 독립적인 계층구조를 갖는 SOA 위에서의 구현 기술로서의 웹서비스를 기본으로 한다. 웹서

비스 기술은 분산 시스템과 소프트웨어 컴포넌트 기반 개발의 CORBA, .NET, J2EE 등의 플랫폼 기술에 웹 프로토콜과 XML기술을 적용하여, 몇 가지 부가효과를 얻는 기술로서 등장하였다.



(그림 1) SOA의 영향요소들

SOA 웹서비스 기술은 약한 결합력과 높은 상호운용성, 그리고 플랫폼 독립성의 특징을 강점으로 분산 컴퓨팅 환경에서 각광 받고 있다. 그러나 상호운용성과 플랫폼 독립성 등의 웹서비스 특징에 지대한 영향을 주는 XML 기술의 메커니즘이 다른 한편으로는 웹서비스들 사이의 반복적인 XML 메세지 교환으로 인해 실행 성능을 낮추는 역효과를 내고 있다. 이와 관련된 연구들이 진행되고 있으며, 본 논문 에서 제안하는 방법도 실행 성능을 향상시키는 효과를 갖는다.

웹서비스에서의 SOAP 메세지 구현과 기존의 Java RMI나 CORA/IIOP와의 수행 성능을 실험을 통해 평가하는 연구[8]에서 비효율적인 SOAP 메세지의 원인이 두 가지로 분석되었다. 하나는 논리적으로는 하나의 일을 하는 메세지 전송이 실제 구현상으로는 여러 번의 메세지 전송이 일어나는 경우들이 존재하는 것과, 또 다른 하나는 XML 파싱과 포맷팅에 관한 것이다. CORBA와의 비교를 통해 이와 유사한 분석 결과가 나오기도 했다[9]. 이러한 문제들을 해결하기 위하여 SOAP 메세지 인코딩과 디코딩을 최적화 방안이 제안되었고, 이것이 웹서비스를 사용하는 비즈니스 어플리케이션의 성능 향상에 기여한다[10]는 연구 결과가 있었으며, 따라서 최근 연구들은 SOAP 메세지 구현에 대한 다양한 최적화 방안들을 내 놓고 있다.

이들은 모두 웹서비스들 사이의 메세지 전송 성능에 초점을 맞추고 있다. 본 연구도 물론 웹서비스 시스템의 성능 향상 효과를 노리는 방안을 제안하고 있으나, 본 연구의 적용 도메인은 유비쿼터스 컴퓨팅의 어플리케이션 구현 모델로서의 웹서비스이며, 이때 가장 고려해야 하는 상황인식을 위한 적응성을 수행하는 부분에서의 성능 향상을 목표로 하고 있다.

2.2 상황인식 적응성 연구

상황정보의 변화에 대한 적응성 부분은 최근 UC 어플리케이션 구현을 연구하는 분야에서 많은 이슈가 되는 부분이다. 상황인식 어플리케이션의 구현 모델로서 제안된 컨텍스트 툴킷[7]은 상황정보 수집 및 추상화 부분과 상황인식 어플리케이션의 분리를 목적으로, 센서에서부터 최종 서비스까지 상황정보 추상화 정도에 따라 각각의 수준을 독립적인 컴포넌트 단위로 구현하는 모델이다. 이는 상황정보를 처리하는 부분과 최종 서비스하는 부분을 계층으로 분리함으로써, 상황정보에 대한 적응을 최상위층만을 고려하여 수행할 수 있는 장점이 있다. 본 연구도 이 개념을 받아들여 UC 어플리케이션의 하위 층이 상황정보를 처리하고 이 정보를 통해 사용자에게 서비스하는 층에 웹서비스를 사용하는 통합 서비스 코드를 두는 모습으로 구현한다.

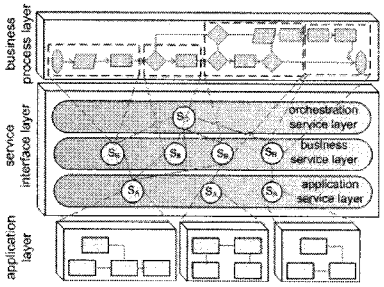
상황정보에 대한 적응성을 컴포넌트 모델을 통해 해결하는 연구 [11,12,13]들이 있다. 이 가운데 Multifacet[12] 접근법은 상황인식 어플리케이션이 상황정보 변화에 적응하는 방안을 제시한다. 이는 중앙에서 어플리케이션 행위들의 수행 순서를 수행 조건과 우선순위 등에 기반을 두어 관리한다. 어플리케이션은 상황정보에 민감한 컴포넌트와 상황정보와 상관없는 컴포넌트로 구성되어진다는 아이디어에서 출발한 방법이다. Gravity [13]는 실행 시간에 서비스 존재의 다양성에 자동으로 적응하는 능력을 갖는 컴포넌트 기반 어플리케이션 구축

을 지원하는 것이다. 동적으로 변화되는 서비스에 적응하는 부분으로서, 응용부분을 유비쿼터스 컴퓨팅에서의 상황정보 변화에 적응하는 것으로 내세우고 있다. Gravity와 Multifacet등은 전체 어플리케이션을 개발할 때 어떻게 하면 상황정보에 민감하게 구현할 수 있을까에 초점을 맞추어 상황정보에 민감해야 하는 부분을 독립적인 컴포넌트로 작성하여 전체 어플리케이션에의 변화를 최소화 시키는데 목적을 가지고 있다. 그에 비하여 본 연구는 상황정보에 민감한 그 컴포넌트들을 각 상황정보에 대하여 대치되어져야 하는 컴포넌트와 어떤 방식으로 연관관계를 설정하는 것이 어플리케이션의 복잡도를 줄여 효율적인 결과를 얻을 수 있을까 하는데 초점을 맞추어, 다형성의 개념을 적용한 접근법을 제공하고 있다.

3. 통합 서비스 구현 모델

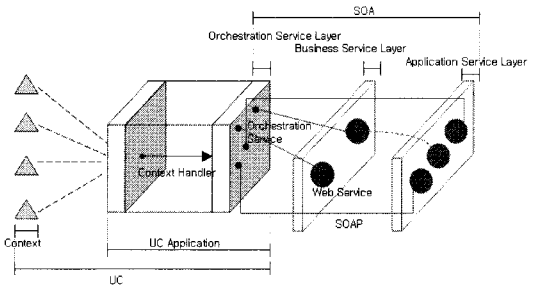
3.1 통합 서비스 계층

웹서비스 구현 기술에서 사용하는 클라이언트 코드는 SOA 관점에서 보면 오케스트레이션 서비스에 해당한다 [7]. UC 디바이스에 들어가는 서비스는 웹서비스 시스템의 클라이언트이고, 이 클라이언트가 동적 SOAP 메세지 전송을 통해 여러 서비스들의 기능을 사용하고 관리한다. 이러한 관점에서 그림2의 SOA 서비스 계층 구성도의 "Orchestration Service Layer"에 존재하는 통합 서비스에 해당된다.



(그림 2) SOA 서비스 계층 구조도 [7]

그림 2는 SOA가 제시하는 이상적인 서비스 인터페이스 계층의 구조를 보여준다. 이상적인 SOA가 되려면 서비스 인터페이스 계층을 가져야 하며, 그 계층은 독립적인 세 개의 서비스 계층으로 이루어져야 한다. 그들은 각각 통합 서비스 계층, 비즈니스 서비스 계층, 어플리케이션 서비스 계층이다. 각각은 구분되는 역할을 갖는 서비스들의 집합이다. 비즈니스 프로세스의 연관된 작업들을 지원하는 비즈니스 서비스들을 위한 계층이 비즈니스 서비스 계층이다. 또한 어플리케이션 오퍼레이션들을 지원하는 웹서비스들을 위한 계층이 어플리케이션 서비스 계층이며, 비즈니스 서비스와 어플리케이션 서비스들에 대한 통합 서비스들이 위한 계층이 통합 서비스 계층이다.



(그림 3) UC와 SOA의 통합 구조도

웹서비스는 UC 디바이스의 통합서비스와 물리적으로 독립된 공간에 존재하고, 변화하는 상황정보에 맞는 웹서비스로 통합서비스가 SOAP 메세지를 전송함으로써, 웹서비스를 실행시킨다. 상황정보를 처리해야 하는 UC 환경과 그에 따라 수행되는 웹서비스들의 SOA 구조는 UC 디바이스 내의 통합서비스를 중심으로 배치될 수 있다. 그림 3은 유비쿼터스의 상황인식 어플리케이션 구현 모델 [14]과 SOA의 서비스계층 구조를 SOA의 통합서비스를 중심으로 배치한 UC에서의 SOA 서비스 구조도이다 [3].

그림 3은 통합서비스를 중심으로 좌측으로는

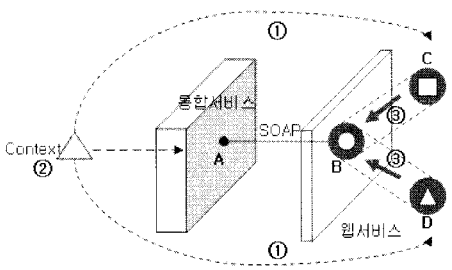
UC 환경에서의 UC 어플리케이션과 상황정보들과의 관계를 보여주고, 우측으로는 SOA 서비스 계층 구조를 보여주고 있다. 서비스 층을 이루는 서비스들 사이에는 직접적인 결합관계가 존재하지 않으며, 통합서비스와 SOAP 메세지를 주고받는 관계로 결합되어져 있다. 또한 상황정보를 받아오는 센서 등의 하드웨어 관련 계층위에 사용자와 가장 가까운 최상위 계층에 우리가 대상으로 하는 통합서비스가 존재하며, 이는 새로운 상황정보를 상황정보 처리기에서 받아 새로운 상황정보에 알맞은 구현을 갖는 우측의 웹서비스를 선택하고 그에게 SOAP 메세지를 전송하여 서비스를 요구하게 된다. 본 장에서는 UC와 웹서비스의 수행 성능 향상의 중요성을 인식하여, 그림 3의 우측 UC의 특성인 상황인식 적응성을 만족하는 통합 서비스와 SOA 서비스들 구현에 다형성을 구현한다. 앞서 1장 서론에서 우리가 해결하고자 하는 문제점인 상황정보 적응성과 수행 성능에 긍정적인 효과를 내도록 다형성을 구현하는 방안을 3.2와 3.3을 통해 기술한다.

3.2 통합 서비스의 적응성 구현

UC 어플리케이션 가운데 하나인 in-car information system [5]은 운전자에게 날씨, 도로교통상황, 지역정보 등을 알려준다. 이때 in-car information system은 자동차 내에 설치되어져 이동한다. 이때 현재의 위치 또는 시간에 따라 다른 행위를 요구된다. 예를 들어 현재 위치가 도심이라면 일반적인 날씨정보를 제공하고, 이동하면서 산악지대에 들어서면 산악지대 안전운행에 도움이 되는 정보를 날씨정보와 함께 제공하도록 한다. 즉, 위치라는 상황정보가 도심에서 산악지대로 변화가 일어나고 이러한 상황정보 변화에 in-car information system은 적응하여야 한다. 적응의 내용으로는 날씨 정보를 산악운행정보와 연계시켜 제공해야 하는 것이다.

본 논문은 이러한 상황정보 적응성을 다형성을 이용하여 해결하려고 한다. 다형성은 동일한 인터페이스를 통하여 다양한 구현내용이 가능하도록 하는 구현모델로서 객체지향 소프트웨어 개발의 하나의 방법으로 사용되고 있다. 동일한 인터페이스를 사용함으로써, 그를 사용하는 클라이언트 입장에서는 구체적인 구현 내용과 무관하게 일관된 인터페이스를 통하여 구현행위를 사용할 수 있다. 따라서 상황정보에 따라 알맞은 행위를 나타내야 하는 경우, 구현내용만 다양하게 유지하고 이를 호출하는 인터페이스를 일관되게 유지하게 된다면, 상황정보 변화에 보다 복잡하지 않게 적응할 수 있다.

다형성을 적용하여 상황정보 적응성을 어떻게 지원하는지를 그림 4로 표현하였다. 동일한 인터페이스를 사용하므로, 새로운 서비스를 끼워 맞춰야 하는 상황정보 적응성 부분에 상황정보에 따라 새로 선택된 웹서비스 C 또는 D는 B와 동일한 인터페이스를 유지하면서 내부의 구현 모습만 사각형, 삼각형으로 구현되어지도록 구현되고, SOAP 메세지를 전송하는 통합서비스의 한 부인 A는 B, C, 또는 D의 겨우 모두 동일한 곳으로 메세지를 전달하고 있다. 그림 4의 C와 D는 기존 웹서비스인 B와 동일한 상황정보를 다르게 처리하게 된다. 이는 "동일한 인터페이스 다양한 구현"인 다형성이 적용된 결과이다. 이때 통합서비스 A에서는 웹서비스 검색부터 시작하여 SOAP 메세지 작성하는 과정 없이, 동일한 이름의 웹서비스로 동일한 구조를 갖는 SOAP를 전송하는 것으로 상황정보에 적응한다.



(그림 4) 다형성을 통한 상황정보 적응성 구현

앞서 언급한 in-car information system으로 그림 4를 설명하면, 그림 4의 B가 도심에 위치했을 때 일반적인 날씨정보를 제공하는 웹서비스가 된다. 이후 산악지대로 이동하면서 그림 4의 ② context에 변화가 생겼고, 그 변화를 예측하여 이미 구현되어져 있는 웹서비스 C가 있다. 물론 그림 4의 ①의 과정으로 C는 B와 동일한 인터페이스를 갖도록 구현되어져 있다. 그러므로 in-car information system의 A부분은 B에게 했듯이 동일한 방법으로 동일한 구조의 SOAP 메세지를 전송하면 이번에는 그림 4의 ③을 통하여 C가 이 메세지를 받아 C의 구현내용이 수행되는 것이다. C는 산악지대에서의 운행을 위한 날씨정보를 제공하게 된다.

3.3 다형성 적용으로 인한 복잡도 감소

하나의 통합서비스에서 상황정보에 따라 서로 다른 웹서비스를 수행해야 하므로 본 논문은 "one interface multiple implementations"를 구현하는 객체지향의 다형성을 이용한다. 객체지향에서 다형성을 구현하는 두 가지 기법이 있다. 오버로딩(Overloading)과 오버라이딩(Overriding)이 그것이다. 두 기법의 구성요소는 메소드의 이름과 매개변수에 있다. 두 기법 모두 이름은 동일하게 유지하도록 하였으나, 매개변수는 다르게 운용하고 있다. 매개변수는 웹서비스 각각에 전달되는 값을 담게 된다. 오버로딩은 동일한 이름과 서로 다른 매개변수를 사용하는 기법이고, 오버라이딩은 동일한 이름과 동일한 매개변수를 통하여 기존 메소드 - 객체지향에서는 상속관계의 부모클래스의 메소드- 를 가리면서 새로운 메소드를 구현하는 기법이다. 두 기법 모두 동일한 이름의 사용을 기본으로 한다. 따라서 새로운 환경을 위한 새로운 웹서비스를 개발하여 배치시킬 때, 통합서비스에서 인지하고 있는 기존의 웹서비스의 URL과 동일하게 만들어야 한다.

만일 오버로딩을 사용하여 구현한다면, 각 웹서비스의 WSDL을 분석하여 SOAP 메세지를 재작성해야 한다. WSDL의 분석이 필요하다는 것은 UDDI 또는 기타 웹서비스 브로커로부터 원하는 웹서비스의 WSDL 위치를 알아내고, 그 위치에서 WSDL을 다운로드받은 후, 이를 분석하여 요구되는 매개변수 스펙을 찾아내야 한다. 이러한 절차가 인간이 하기에는 어렵지 않으나, 상황인식 어플리케이션의 경우, 통합서비스 수행 중에 일어나야 하므로 모두 코드로 구현되어져야 한다. 통합서비스내의 절차가 많아질수록 수행 복잡도는 증가한다. 만일 오버라이딩을 사용하여 구현한다면, 적응을 위해 대치하기 위한 웹서비스 구현의 WSDL을 동일하게 사용하게 된다. 앞서 오버로딩의 경우에 WSDL내의 오퍼레이션등의 이름들을 다시 찾아내야 하는데 반해, 오버라이딩의 경우에는 한번 찾아진 오퍼레이션등에 관한 정보가 재사용될 수 있다. 그 이유는 오버라이딩이 인터페이스의 이름뿐 아니라 그에 전달되는 파라미터까지도 동일하게 유지해야 하기 때문에 동일한 WSDL을 그대로 사용하기 때문이다.

본 논문에서 고려대상으로 하고 있는 상황정보 적응성과 성능, 그리고 웹서비스 개발 입장과 이들 서비스를 사용하는 통합서비스 구현 입장 등 4가지 측면에서 오버로딩과 오버라이딩을 표 1에서 분석하였다. 우수성 부분에서 <<는 <보다 큰 차이로 우측이 좌측보다 우세함을 표현하였다. '수행성능' 면에서 보면 오버로딩보다는 오버라이딩이 우수하고, 웹서비스 재사용이나 서비스 구현의 자율성으로 인해 '웹서비스 개발' 측면에서는 오버로딩이 상대적으로 우수함을 표1에서 알 수 있다.

(표 1) 오버로딩과 오버라이딩의 우수성 비교

	오버로딩	우수성	오버라이딩
상황정보 적응성	구현가능	=	구현가능
웹서비스 개발	제약적음	>	제약많음
통합 서비스 구현	복잡	<<	매우단순
수행 성능	적응과정 복잡	<<	적응코드 없음

표 1의 '통합 서비스 구현' 항목에서 오버로딩으로 구현하는 경우 오버라이딩에 비하여 많은 차이로 복잡하다. 즉, 통합서비스 구현 메커니즘 상, 인터페이스의 이름만 맞추는 것은 코드를 줄이는데 큰 의미가 없다. 왜냐하면 이름외의 것을 알아내기 위한 WSDL분석을 반드시 하게 되므로, WSDL 분석과 SOAP 재작성을 피하려고 이용한 다형성 적용의 효과가 희미해진다. 이에 대한 좀 더 자세한 분석이 4장에 있다. 결과적으로 표 1에서 보면 오버라이딩이 오버로딩보다 수행 성능 입장에서 월등히 우월할 수 있으며, 이 결과는 통합서비스 구현에서의 코드 단순화가 주요 원인이 된다.

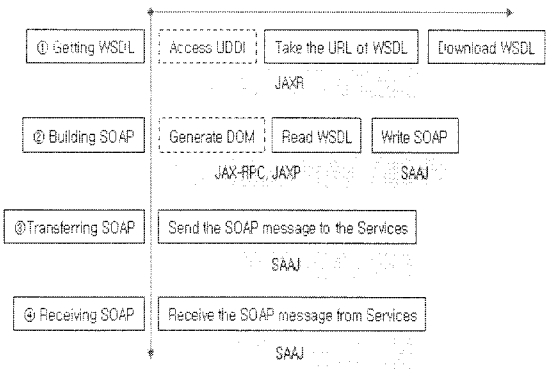
4. 분석

본 논문이 제안한 다형성을 적용하지 않은 경우와 적용하는 경우를 비교하여, 본 제안 방법이 절차를 간소화시킴으로써, 수행 성능에 기여함을 보인다.

4.1 상황정보 적응성 성능 비교

웹서비스 구현 환경으로 J2EE와 .NET이 대표적이다. 그러나 성능 측면에서는 J2EE 환경이 우세하다 [15]. 그리고 웹서비스의 통합서비스, 즉 클라이언트 코드 구현 모델로는 정적 스텁 호출 모델, 동적 프록시(Dynamic Proxy) 호출 모델 , 그리고 동적 호출 인터페이스(Dynamic Invocation Interface) 모델- 이 있다. 클라이언트 코드가 수행 중에 변화하는 웹서비스 정보를 통해 실시간으로 서로 다른 메세지를 보내야 하는 UC의 요구사항을 위해서는 위 세 가지 구현 모델 가운데 동적 호출 인터페이스 모델이 적합하다. 따라서 본 장에서는 J2EE에서 동적 호출 인터페이스 모델로 구현되는 웹서비스 시스템이 상황정보에 대한 적응을 수행하는 통합서비스를 살펴보고, 다형성을 적용하는 경우와 아닌 경우, 그리고 다형성 가운데 오버로딩의 경우와 오버라이딩의 경우를 각각 비교하여 분석한다.

웹서비스를 찾아 해당 WSDL을 가져온 후, 이를 분석하여 SOAP 메세지를 구성하고, 메세지를 전송하고 결과를 받는 일련의 과정이 모두 통합서비스 내에 구현되어지는 경우, 아래 그림과 같은 세부 절차들을 포함하고 있으며, 각각의 세부 절차마다 사용해야 하는 API들은 그림 5와 같다. J2EE 콘테이너와 서버 이외에 JAXR, JAX-RPC, JAXP, SAAJ 등의 여러 API들이 호출된다.



(그림 5) 웹서비스 이용 절차와 관련 API들

기존의 웹서비스를 사용하는 클라이언트 개발에서는 위 그림의 ① 과정은 이미 매뉴얼하게 진행되었고, 그를 통해 필요한 SOAP 메세지도 거의 완성시켜놓고, 실제 전달할 값에 대해서만 변형을 주어, 결정되어져 있는 웹서비스로 보내고 수행결과를 받는 작업(그림 5의 ③~④) 위주로 구성되어진다. 그러나 상황정보에 적응하려면, 수행할 웹서비스의 구현내용이 변화될 필요가 있으며, 새로운 웹서비스의 감지부터 적합한 SOAP 메세지 구성까지 모두(그림 5의 ①~④) 클라이언트 측의 코드 내에 구현되어져야 한다.

(표 2) 다형성 여부에 따른 웹서비스 이용 절차 비교

단계		가중치			
Getting WSDL	*Access UDDI (optional)*	5	○	×	×
	Take the URL of WSDL	1	○	×	×
	Download WSDL	1	○	○	×
Building SOAP	*Generate DOM (Optional)*	3	○	○	×
	Read WSDL	2	○	○	×
	Write SOAP	8	○	△(2)	△(2)
Transferring SOAP	*Send the SOAP to the Services*	0.5	○	○	○
Receiving SOAP	*Receive the SOAP form Services*	0.5	○	○	○

위의 각 단계들을 오버로딩과 오버라이딩을 적용했을 경우, 표 2와 같이 절차가 줄어드는 것을 볼 수 있다. 표 2에서 회색 셀이 각 경우 요구되는 단계들의 영역이므로, "다형성>오버로딩>오버라이딩"의 순으로 요구되는 단계가 적음을 알 수 있다. 표에서 단순한 하나의 단계일지라도 이를 API들의 메소드들을 하나씩 호출하면서 수행해야 하므로 많은 코드 라인이 요구되므로, 위 표에서 보이는 차이는 실제 코드 중심의 절차수로 분석할 경우 보다 격차가 크게 나타난다. 이를 계산하기 위하여 각 단계의 알고리즘과 코드 [16]를 분석하고 절차의 수를 계산하여 표 2의 가중치 값을 얻었다.

"Access UDDI"는 UDDI를 사용하지 않을 경우에는 필요없는 부분으로서, 웹서비스 획득 단계에서 선택적으로 사용되는 과정이다. 또한 "Generate DOM"은 DOM트리 구성없이도 XML파일의 내용을 꺼내어올 수 있으므로 DOM트리 생성에 대한 부분을 선택적으로 사용되는 부분으로 정의하였다. 각 가중치의 값들은 아래의 그림 6과 같이 해당 단계를 구현한 소스코드의 주요 스텝들의 수를 세어 가중치 값으로 사용하였다. 예를 들어, 그림 5의 ②에 속하는 "Write SOAP"의 경우, 그림 6의 8단계의 일이 구현 코드 상에 수행된다. 아래와 같으므로, "Write SOAP"의 가중치는 8로 결정된다. 표 2에서 오버라이딩이나 오버로딩의 개념을 구현할 경우, 그림 6의 8개 단계 가운데 마지막 두 단계(그림 6의 7과 8)만이 동적 적응을 위하여 지속적으로 반복되므로 표 2에 △로 표기하였으며, 이때 가중치는 2가 된다. 각각의 경우 상황정보 적응성을 동적으로 해결하기 위하여 요구되는 단계들의 총 가중치 값은 표 2에서 보듯이 각각 21, 9, 3으로 분석되었다.

WriteSOAP

1. SOAPMessage 객체 생성
2. SOAPPart 객체 생성
3. SOAPEnvelop 객체 생성
4. SOAPFactory 객체 생성
5. SOAPBody 객체 얻기
6. 바디 엔트리 생성
7. 바디 엔트리 자식 엘리먼트 생성
8. SOAPConnection 객체 생성

(그림 6) "WriteSOAP"구현의 주요 절차들

각 절차의 수는 소프트웨어 복잡도 측정 메트릭 가운데 하나인 LOC와 연관된다. LOC는 프로그래밍의 명령어 수를 세어 높은 LOC를 갖는 소프트웨어에 대하여 높은 복잡도를 갖는 것으로 판단하는 소프트웨어 공학 분야의 고전적인 메트릭이다. 표 1에서 사용된 가중치는 각 단계에 대한 LOC를 측정한 것이다. 소프트웨어 복잡도는 소프트웨어 구현 및 시험 비용 증가를 유도하는 소프트웨어 품질 가운데 하나로서, 웹서비스에도 동일한 의미를 갖을 수 있다.

각각의 가중치 값은 상황정보의 변화가 일어나고 그에 적응해야 하는 횟수가 증가함에 따라 비례하여 그 차이도 커진다. 예를 들어 세 번의 상황정보 변화로 인해 세 번의 상황정보에 대한 적응 코드가 수행되어져야 한다면, 각각의 경우는 21*3, 9*3, 3*3의 가중치를 갖게 된다. 이에 따라

그들 사이의 차이는 더욱 커지게 된다. 3장에서 언급한대로 오버로딩보다는 오버라이딩이 적은 가중치를 요구하고 있으며, 따라서 오버라이딩을 이용한 다형성을 구현하면 "w/o 다형성"에 비하면 획기적인 비용 절감 효과를 기대할 수 있다. 만일 상황정보 변화가 3번 일어나고 그에 따른 적응과정을 3번 반복해야 한다면, 표2의 각 단계들을 세 번 반복하는 결과로서, "w/o 다형성"과 오버라이딩은 (21-3)*3 의 가중치 차이가 나며, "w/o 다형성"과 오버로딩의 차이는 (21-9)*3의 차이로 다형성을 사용하는 경우가 더 효율적이라고 할 수 있다. 이러한 차이는 상황정보 변화에 적응하는 횟수가 증가할수록 비례하여 커지게 된다.

4.2 웹서비스 개발에서의 취약점

본 논문에서 제안하는 다형성을 이용하여 통합 서비스를 구현하는 방안은, 결과적으로 상황정보 변화에 따라 반복적으로 수행해야 하는 통합 서비스 코드의 크기를 줄여주어 수행 시간 성능을 높이는 효과가 있다. 그러나 앞서 3장에서 언급했듯이, 일단 다형성을 사용한다는 의미가 인터페이스를 고정 시켜놓는다는 의미를 가지므로, 이 부분이 오히려 웹서비스를 개발하는 측면에서 제약이 된다. 지정된 이름과 지정된 파라미터를 유지해야 하므로 개발의 자율성에 저해된다. 이는 기존의 웹서비스 재사용을 어렵게 한다. 왜냐하면 기존의 웹서비스들은 다형성을 위해 맞추어져야 하는 이름과 파라미터에 대한 정보는 갖고 있지 않기 때문이다. 다형성을 위하여 약속된 이름과 파라미터에 대한 정보가 먼저 정의되고, 그에 따라 작성된 웹서비스들이 본 논문에서 제안하는 방법으로 운용될 수 있다. 즉, 다형성 규칙을 엄밀하게 적용시킬수록 통합 서비스 코드의 크기는 줄어들지만, 그에 따른 웹서비스 사용의 자율성은 줄어들게 된다. 따라서 웹서비스 개발의 자율성을 일정부분 보장하고자 한다면, 다형성의 기법 가운데 오버로딩을 사용하여 해결할 수 있다. 표 1에서 분석했듯이, 다형성의 두 가지 기법 가운데 오버로딩이 오버라이딩보다 웹서비스 개발에 요구하는 제약사항이 적다.

5. 결론

본 논문은 상황인식 어플리케이션의 구현을 웹서비스로 하는 경우, 상황정보 변화에 적응해야 하는 적응성의 효율성을 지원하기 위하여, 웹서비스 통합 서비스 구현에 객체지향의 다형을 개념을 적용하여 보다 적은 크기의 적응성 관련 코드를 생성하는 방안을 제안하였다. 시간, 장소, 행위자 등의 어플리케이션 상황정보들은 끊임없이 변화를 거듭하고, 이렇게 변화하는 상황정보에 맞추어 어플리케이션의 행위가 변화되어져야 하기 때문이다. 따라서 상황인식 어플리케이션에서의 상황정보 적응성은 매우 중요한 부분이다. 이때 발생 가능한 모든 상황정보를 위한 코드들을 하나의 어플리케이션에 담는 것을 비용 면에서 비효율적이다. 또한 HP 쿨타운과 같이 어플리케이션들은 웹 환경을 기반으로 하는 네트워킹을 기본으로 한다. 따라서 구현의 실제 모습을 웹서비스로 하는 어플리케이션의 사용이 적합하다.

본 논문은, 컴포넌트를 사용하는 기존의 적응성 해결 방안과는 달리, 객체지향의 다형성 기법을 이용하여 적응성 문제를 해결하면서, LOC 소프트웨어 복잡도 메트릭에 근거하여 적응성 코드의 사이즈를 줄이는 것을 목적으로 한다. 상황인식을 위해 다양한 서비스 구현을 번갈아가면서 사용해야 경우, 다형성을 적용하게 되면 적응을 위해 대체되어지는 서비스들 사이에 WSDL을 공유하는 효과를 낸다. 결국 특정 서비스 사용을 위해 WSDL을 습득하고 분석하는 과정이 생략될 수 있으므로, 적응을 위한 서비스 대체 과정이 단순하게 되는 것이다. 이를 통하여 적응성 수행 시간 비용이 줄어드는 효과를 볼 수 있다. 통합서비스

가 현재의 상황정보에 적합한 웹서비스를 UDDI 또는 개인적인 저장소에서 찾고, 그의 WSDL 분석을 통해 SOAP 메세지를 작성하여 해당 웹서비스에게 전달하는 일을 수행하도록 코드화해야 한다. 물론 다양한 API들이 존재하여 이들 전 과정에 대한 자동화가 가능한 코드를 구현할 수 있다. 앞서 기술한 일련의 과정들을 상황정보의 변화가 있을 때마다 항상 수행해야 하므로 웹서비스의 SOAP 메세지 전송으로 인해 근본적으로 갖는 성능상의 문제점과 더불어 웹서비스 시스템은 수행성능이 떨어지게 된다. 본 논문에서는 이를 해결하는 방안으로 객체지향의 다형성 원리를 통합 서비스 구현에 적용하는 방안을 제안하였다.

실제 웹서비스 통합 서비스를 다형성의 두 가지 기법으로 각각 구현하는 경우와 기존의 방법으로 구현하는 경우에 대한 사례 분석을 통하여, 각 경우에 어느 정도의 코드 사이즈가 줄어드는 효과가 있는지를 보였다. 분석 결과 표 2에서 보듯이 다형성 적용이 있는 경우와 없는 경우 확연한 수행 절차수의 차이를 볼 수 있으며, 상황정보에 대한 적응성 측면에서는 우리의 다형성이 효과적임을 설명하였다. 그러나 일반적인 웹서비스 시스템이 갖는 장점 가운데 하나인 서비스 재사용이 어려워지고, 웹서비스 개발자가 통합 서비스를 개발할 때, 정해진 규칙에 따라 구현을 해야 하는 개발의 경직성이 단점으로 분석되었다. 그러나 본 논문이 제안한 방법은 상황인식 어플리케이션 개발에 초점을 맞추었으므로, 상황인식 어플리케이션 개발에 웹서비스 구현 기술을 이용하는 경우 사용 가능하다. 제안한 방안에 따르는 통합 서비스 구현을 통하여 어플리케이션을 구축할 경우, 상황인식 어플리케이션의 적응성을 비교적 짧은 절차를 통해 구현할 수 있다.

참 고 문 헌

[1] John Barton, Tim Kindberg, "The Cooltown User Experience," HP Technical Report, 2001

[2] Gartner Group, "Gartner's 2006 Emerging Technologies Hype Cycle Highlights Key Technology Themes," http://www.gartner.com, 2005

[3] Hoijin Yoon, "A Convergence of Context-Awareness and Service-Orientation in Ubiquitous Computing," International Journal of Computer Science and Network Security, Vol.7 No.3, March 2007

[4] Gregory D. Abowd, "Software Engineering Issues for Ubiquitous Computing," Proceedings of ICSE '99, pp.75-84, May 1999, LA, CA, USA

[5] Ian Sommerville, 'Software Engineering', 8th edition, Addison Wesley, 2007

[6] Wei Jun et al, "Speed-up SOAP Processing by Data Mapping Template," Proceedings of the 2006 International workshop on Service-oriented Software Engineering, 2006

[7] Thomas Erl, 'Service Oriented Architecture - Concepts', Prentice Hall, 2005

[8] D.David and M.Parashar, "Latency Performance of SOAP implementations," Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid," pp.407-412, 2002

[9] R.Elfwing, U.Paulsson, and L.Lunberg, "Performance of SOAP in Web Service Environment Compared to CORBA," Proceedings of the 9th Asia-Pacific Software Engineering Conference, 2002

[10] K.Chiu, M.Govindaraju, and R.Bramley, "Investigating the limits of SOAP performance for scientific computing," Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pp.246-254, 2002

[11] Hoijin Yoon, Byoungju Choi, "The Context Driven Component Supporting the Context Adaptation and the Content Extension," Journal of Information Science and Engineering, Vol. 22 No.6 pp.1485-1504, 2006

[12] Anca Rarau, Kalman Pusztai, and Ioan Salomie, "MultiFacet Item Based Context-Aware Applications," International Journal of Computing & Information Sciences, Vol. 3, No. 2, 2005, pp.10-18.

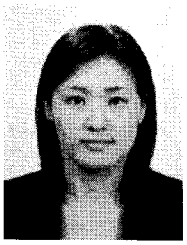
[13] Humberto Cervantes and Richard S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model," in Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004.

[14] Karen Henricksen and Jadwiga Indulska, "Developing context-aware pervasive computing applications : Model and approach," Pervasive and Mobile Computing, Vol.2, No.1, pp.37-64

[15] Sun Microsystems Inc. Web Services Performance comparing Java2 Enterprise Edition and .NET Framework, http://java.sun.com/performance/reference/whitepapers/WS_Test-1_0.pdf, 2004

[16] 신민철, 'XML 웹서비스," 프리렉, 2004

◐ 저 자 소 개 ◑



윤 회 진(Yoon Hoi-jin)
1993년 이화여자대학교 전자계산학과 (이학사)
1998년 이화여자대학교 대학원 컴퓨터학과 (이학석사)
2004년 이화여자대학교 대학원 컴퓨터학과 (공학박사)
2005년~2007년 이화여자대학교 컴퓨터학과 전임강사
2007년~현재 협성대학교 컴퓨터공학과 전임강사
관심분야 : 소프트웨어테스트, 서비스지향아키텍쳐, 뮤테이션테스팅,
E-mail : hjyoon@uhs.ac.kr