DOI: 10.3745/KIPSTD.2008.15-D.2.223

# 자율적 피드백 기반 웹 서비스 선정을 위한 품질 브로커 아키텍처의 설계 및 구현

서 영 준[†] · 송 영 재[††]


## 요 약

최근 웹 서비스 분야는 기업내외의 효율적인 통합 환경을 제공해 주면서 웹 서비스의 도입을 원하는 업체가 증가하고 있다. 또한 웹 서비스가 발전하면서 새로운 비즈니스 모델이 등장하고, 웹 서비스로 인해 국내 기업 환경 및 e-비즈니스 환경이 변화하고 있다. 유사한 기능을 제공하는 웹 서비스가 증가함에 따라 사용자의 요구에 가장 적합한 서비스를 찾는 방법이 더욱 중요시 되고 있다. 많은 유사한 웹 서비스들 가운데 하나를 선택해야 할 때, 서비스 사용자는 일반적으로 웹 서비스의 품질 정보를 필요로 하게 된다.

그러나 웹 서비스의 광고 QoS 정보는 항상 신뢰성이 있지는 않다는 문제점이 있다. 서비스 제공자가 더 많은 사용자들을 끌어들이기 위해 부정확한 QoS 정보를 게시하거나, 게시된 QoS 정보가 오래 되었을 수도 있다. 따라서 현재의 사용자들이 웹 서비스로부터 받는 QoS를 평가하고 이러한 평가를 공유하는 중개자의 존재는 새로운 사용자들에게 가치 있는 정보를 제공할 수 있다.

본 논문에서는 서비스 사용자의 입장에서 사용자가 원하는 최적의 품질을 제공하는 서비스를 찾도록 도와주는 에이전트 기반 품질 브로커 아키텍처를 제안한다. 동적으로 웹 서비스를 선정하는 아키텍처를 사용자에 제공함으로서 사용자의 품질 요구 변경 문제를 해결할 수 있다. 즉 사용자는 품질 브로커 서버에 연결된 UDDI 브라우저를 통해 최적의 품질 척도를 제공하는 서비스를 검색할 수 있다. 또한 각 서비스의 품질 척도 값 결정에는 사용자 개입이 최대한 배제된다. 기존 선정 아키텍처에서는 사용자의 주관적 서비스 등급 선정으로 객관적 평가가 어려웠으나, 에이전트가 사용자 위치에서 모니터링 한 바인딩 정보를 통한 품질 척도 값 결정으로 객관성을 확보할 수 있다. 즉, 제공자들이 제공하지 못하는 서비스의 QoS 정보를 사용자측 에이전트들의 피드백으로 인한 QoS 정보 공유로 해결하고자 한다.


**키워드 : 웹 서비스, 품질 브로커, 품질 모니터링, 품질 측정, 이동 에이전트**

# Design and Implementation of Quality Broker Architecture to Web Service Selection based on Autonomic Feedback

Young-Jun Seo[†] · Young-Jae Song[††]


## ABSTRACT

Recently the web service area provides the efficient integrated environment of the internal and external of corporation and enterprise that wants the introduction of it is increasing. Also the web service develops and the new business model appears, the domestic enterprise environment and e-business environment are changing caused by web service. The web service which provides the similar function increases, most the method which searches the suitable service in demand of the user is more considered seriously. When it needs to choose one among the similar web services, service consumer generally needs quality information of web service.

The problem, however, is that the advertised QoS information of a web service is not always trustworthy. A service provider may publish inaccurate QoS information to attract more customers, or the published QoS information may be out of date. Allowing current customers to rate the QoS they receive from a web service, and making these ratings public, can provide new customers with valuable information on how to rank services.

This paper suggests the agent-based quality broker architecture which helps to find a service providing the optimum quality that the consumer needs in a position of service consumer. It is able to solve problem which modify quality requirements of the consumer from providing the architecture it selects a web service to consumer dynamically. Namely, the consumer is able to search the service which provides the optimal quality criteria through UDDI browser which is connected in quality broker server. To quality criteria value decision of each service the user intervention is excluded the maximum. In the existing selection architecture, the objective evaluation was difficult in subjective class of service selecting of the consumer. But the proposal architecture is able to secure an objectivity with the quality criteria value decision where the agent monitors binding information in consumer location. Namely, it solves QoS information of service which provider does not provide with QoS information sharing which is caused by with feedback of consumer side agents.


**Key Words : Web Service, Quality Broker, Quality Monitoring, Quality Measurement, Mobile Agent**

---

※ 이 연구는 2006년도 경희대학교 연구비지원에 의한 결과임.(KHU-20060585)
† 정 회 원 : 국가기록원 대통령기록관 공업연구사
†† 종신회원 : 경희대학교 전자정보학부 교수(교신저자)
논문접수 : 2007년 11월 5일, 심사완료 : 2007년 12월 14일

## 1. 서 론

최근 IT 분야의 최대 화두가 되고 있는 웹 서비스는 기업내외의 효율적인 통합 환경을 제공해 주면서 웹 서비스의 도입을 원하거나, 도입 중인 업체가 증가하고 있다. 또한 웹 서비스가 발전하면서 새로운 비즈니스 모델이 등장하고, 웹 서비스로 인해 국내 기업 환경 및 e-비즈니스 환경이 변화하고 있다.

그러나 웹 서비스가 활성화되기 시작하면서 해결해야 할 문제점들 또한 증가하고 있다. 이러한 문제들 중에서 초점이 되고 있는 요인은 웹 서비스 선정 문제이다. 현재 서비스 사용자들은 기능적 요구를 만족하는 웹 서비스를 찾기 위해 UDDI 레지스트리를 수작업으로 검색한다. 만약 적합한 서비스가 발견된다면, QoS 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].

웹 서비스가 널리 보급될수록 웹 서비스의 서비스 품질이 특정 웹 서비스의 효용성과 인기도에 직접적인 영향을 주기 때문에 차별화 포인트로서 더욱 중요해 지고 있다. 비즈니스 로직을 웹 서비스 형태로 공개하는 것 자체는 그다지 어려운 일은 아니나, 웹 서비스가 잠재적인 고객들의 요구에 맞는 서비스를 제공할 수 있도록 설계하는 부분이 어려운 일이다. 많은 유사한 웹 서비스들 가운데 하나를 선택해야 할 때, 서비스 사용자는 일반적으로 웹 서비스의 품질 정보를 필요로 하게 된다. 비록 UDDI가 서비스 품질 정보를 제공하도록 설계된 것은 아니지만, UDDI 레지스트리들은 서비스 사용자들에게 편의를 제공하기 위해서 이러한 정보들을 포함하려 하는 추세이다. 또한, 애플리케이션이 다양한 웹 서비스들로 조립되는 시나리오에서는 사용자들이 웹상에서 복잡한 질의들을 처리해야 하는 어려움이 있다. 시간 낭비와 어려운 프로세스는 무시하더라도, 사용자는 더 나은 서비스 선택의 기회를 잃기 쉽다[2]. 따라서 웹 서비스 선정 문제를 해결하기 위해서는 서비스 사용자들 간에 품질 속성들을 공유할 수 있는 수단이 필요하다. 공유된 품질 속성들을 사용하여, 서비스 사용자들은 필요한 서비스 구현들의 품질 평판을 알 수 있다.

본 논문에서는 서비스 사용자들을 대신하여 최적의 품질을 제공하는 웹 서비스를 선정하는 에이전트 기반 품질 브로커 아키텍처를 제안한다. 품질 브로커 아키텍처에서는 WSLA 또는 WS-Policy 명세와 같은 복잡한 모델 대신 WSDL과 UDDI와 같은 웹 서비스 표준 기술을 사용하는 기본 웹 서비스 프로토콜을 이용해서도 사용자의 QoS 요구를 만족하기 위한 동적 웹 서비스 발견이 실현될 수 있는 방법을 제안한다. 또한 품질 브로커 아키텍처에서는 서비스 성능에 대한 품질 데이터를 동적으로 확보하기 위한 과정을 소개한다. 모니터링 에이전트는 사용자 측에서 원격 생성되어 서비스 바인딩시 발생하는 정보들을 수집하여 품질 브로커에 전달한다. 품질 브로커는 전달된 바인딩 정보를 저장하고 사용자로부터 QoS 요구가 있을 시에 바인딩 정보를 측정하여 각 서비스의 품질 척도 값을 알려줄 책임이 있다.

현재의 UDDI 레지스트리는 사용자의 기능적 요구를 만족하는 웹 서비스를 찾기 위해 오직 키워드 기반 검색을 제공한다. 또한 선정 프로세스는 전형적으로 설계 시간에 이루어지고 웹 서비스 선택은 정적이고 실행 시간 동안 변경되지 않는다.

본 논문에서는 동적으로 웹 서비스를 선정하는 아키텍처를 사용자에 제공함으로서 실행 시간에 최적의 서비스를 발견하는 프로세스를 제안한다. 즉 사용자는 품질 브로커 서버에 연결된 UDDI 브라우저를 통해 최적의 품질 척도를 제공하는 서비스를 검색할 수 있다. 또한 각 서비스의 품질 척도 값 결정에는 사용자 개입이 최대한 배제된다. 기존 선정 아키텍처[3]에서는 사용자의 주관적 서비스 등급 선정으로 객관적 평가가 어려웠으나, 에이전트가 사용자 위치에서 모니터링 한 바인딩 정보를 통한 품질 척도 값 결정으로 객관성을 확보할 수 있다. 즉, 제공자들이 제공하지 못하는 서비스의 QoS 정보를 사용자측 에이전트들의 피드백으로 인한 QoS 정보 공유로 해결하고자 한다.

그러나 광고 QoS 정보가 없는 서비스 선정 프로세스의 주요 문제는 기아 상태(starvation)이다. 광고 QoS 정보가 없는 상황에서 아직 QoS 정보가 없는 신규 서비스나 초기 선정 당시에 낮은 성능을 제공한 서비스의 경우 설사 높은 성능을 제공하더라도 선정 순위에서 계속적으로 낮은 순위를 받게 된다. 따라서 낮은 순위를 배정받은 서비스들은 무한히 선정되지 못하는 기아 상태가 발생한다. 낮은 순위의 서비스들이 무한하게 선정되지 못하는 문제에 대한 해결은 이동 에이전트의 기능 확대에 있다. 이동 에이전트의 성능 모니터링을 선정된 서비스 외의 등록된 서비스들로 확대함으로서 결과적으로 순위가 증가하게 되어 선정 가능성을 높인다.

본 논문에서 최적의 웹 서비스를 자율적으로 선정하는 선정 알고리즘과 품질 브로커를 설계 및 구현하기 위한 연구 방법은 다음과 같다.

첫째, 서비스 선정을 수행하는 품질 브로커의 아키텍처와 설계 명세를 정의한다. 본 논문에서는 에이전트 기반 품질 브로커 시스템의 아키텍처를 정의하고, UML 다이어그램을 통해 서비스 사용자의 시스템 요구사항을 정의한다. 시스템에 요구되는 것이 무엇인지 기록하기 위해 유스케이스, 클래스 다이어그램 등과 같은 여러 타입의 분석 단계의 모델들이 생성된다. 또한 설계 단계에서는 품질 브로커 시스템의 각 기능별 모델링을 통해 서비스 선정 프로세스의 일련의 과정에 대해서 모델링 한다.

둘째, 서비스 사용자들로부터 많은 양의 품질 정보를

모아 오기 위해 이동 에이전트를 사용한다. 이동 에이전트는 네트워크가 끊길 위험 없이 각 서비스 사용자에게서 성능 정보를 얻어 처리한 다음 자신을 보낸 브로커 서버로 되돌아온다. 이동 에이전트를 사용하면 원격 메소드 호출과 같은 기존의 방법에 비해 효율과 신뢰성을 동시에 높일 수 있다.

셋째, 사용자 위치에서 웹 서비스의 사용자가 인식하는 웹 서비스의 성능을 파악하는데 초점을 맞춘다. 웹 서비스의 성능 측정은 크게 사용자 위치와 제공자 위치에서의 성능 측정으로 나누어진다. 사용자 위치에서의 성능 측정은 사용자 지점에서 웹 서비스의 성능을 측정하는 것이며, 제공 받은 웹 서비스에 대해 사용자가 인식하는 성능을 반영한다. 제공자 위치에서의 성능 측정은 웹 서비스 제공자 내부에서 웹 서비스의 성능을 측정한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 웹 서비스 선정 아키텍처에 대한 이론적 배경을 소개하고, 3장에서는 서비스 사용자의 요구 사항을 이해하고, 전체 품질 브로커 시스템 아키텍처에 대한 분석과 설계 과정을 통해 세부 기능별로 각 구성 모듈의 역할 및 수행 과정을 UML 다이어그램을 사용해 기술한다. 4장에서는 기존 웹 서비스 선정 아키텍처 연구와의 비교 평가를 기술한다. 5장에서는 결론 및 향후 연구 방향에 대해 기술한다.

## 2. 관련 연구

본 장에서는 웹 서비스 선정 아키텍처에 관한 기존 연구들에 대하여 소개한다.

### 2.1 웹 서비스 선정 아키텍처

웹 서비스 선정 아키텍처에 관한 연구는 여전히 초기 연구에 머물러 있으나, 그 필요성 및 중요성은 매우 커지고 있다. 현재 많은 연구 활동들이 사용자의 요구에 가장 만족하는 서비스들을 찾기 위해 서비스 발견 프로세스에 QoS 정보를 고려하는 방법에 관해서 진행되고 있다. 그러한 연구들 중에 일부는 QoS 정보를 지원하기 위해 UDDI 레지스트리[4][5]를 확장하는데 초점을 맞추고 있다[6][7].

A. Shaikhali[7]의 연구에서 UDDIe 프로젝트는 QoS 정보와 같은 서비스와 연관된 사용자 정의 속성들을 기록하고 이러한 속성들에 기반해 서비스 발견을 하기 위해 블루 페이지(blue page)를 추가함으로서 현재의 UDDI 레지스트리를 확장한다. S. Ran[6]의 연구에서는 UDDIe와 유사하게 UDDI 모델에 QoS 증명자(Certifier)로 불리는 새로운 역할을 추가함으로서 웹 서비스 발견을 위한 확장된 모델을 제안하였다. 증명자는 서비스 등록 이전에 웹 서비스의 광고 QoS를 검증한다. 사용자도 웹 서비스 바인딩 이전에 증명자를 사용해 광고 QoS를 검증할 수 있다. 이 시스템은 서비스 제공자들이 등록 단계에서 유효하지 않은 QoS 요구를 게시하는 것을 방지할 수 있으며 사용자들이 서비스 제공자와 만족하는 거래를 보장하도록 QoS 요구를 검증하도록 도와준다. 그러나 이 모델은 선정 알고리즘을 제공하지 않으며, 서비스 발견 프로세스에 사용자의 피드백을 반영하지 않는다. 위의 Shaikhali와 Ran의 연구 모두 서비스 검색과 선정은 여전히 사용자에 의해 이루어진다. 만약 검색된 수천 개의 서비스 중에 적합한 서비스를 선택해야 하는 상황에서는 결코 바람직하지 않다.

일부 연구 프로젝트는 QoS 정보를 다루기 위해 클라이언트와 서비스 제공자들 사이에 QoS 브로커들을 추가하였다[8]. P. Farkas[8]의 연구에서는 UDDI에 QoS aware 서비스를 발견하기 위한 QoS 브로커를 추가함으로서 QoS-enabled 웹 서비스를 제공하는 소프트웨어 아키텍처를 제안하였다. QoS 브로커는 처음에 클라이언트의 기능적 요구에 기반하여 UDDI에서 서비스 발견과 선정을 수행한다. 그다음 QoS 정보를 얻기 위해 선택된 서버에 접속하고 클라이언트의 QoS 요구에 따라 최종 결정을 한다. 그러나 어떻게 설계되었고 제공 기능과 같은 QoS 브로커에 관한 상세한 정보는 없다.

또 다른 연구에서는 클라이언트와 서비스 제공자들 사이에 서비스 선정을 수행하는 에이전트 서버를 추가하였다[1]. E.M. Maximilien[1]의 연구에서는 동적 웹 서비스 선정을 위한 에이전트 프레임워크와 온톨로지를 제안하였다. 서비스 기반 클라이언트 애플리케이션들은 동적으로 에이전트에 의해 구성된다. 클라이언트 애플리케이션이 서비스를 사용할 필요가 있을 때 서비스마다 에이전트들을 생성한다. 서비스들에 대한 QoS 데이터는 에이전트들로부터 수집되며, 에이전시를 통해 에이전트들 사이에 공유된다. 이러한 에이전트 기반 프레임워크는 Web Services Agent Framework(WSAF)로 구현되며, 가장 일반적인 품질 개념들을 정의한 QoS 온톨로지가 정의되었다. 비록 위의 연구들이 QoS를 사용한 서비스 발견의 일부 형태를 기술하나, 어떠한 연구도 사용자들로부터의 피드백을 고려하지 않았다. 서비스 발견과 선정의 결과는 오직 광고된 QoS에 기반을 두나 유효하지 않을 수도 있다. 단 Ran에 의해 제안된 모델에서는 Certifier에 의해 광고된 QoS가 검증된다.

마지막으로 Z. Xu[3]의 연구에서는 에이전트가 서비스의 기능, QoS, 평판 점수에 기반하여 서비스 선정을 수행하는 서비스 발견 모델을 제안하였다. Z. Xu의 연구에서는 UDDI 레지스트리는 QoS 정보를 사용하도록 개선되었으며, 전통적 웹 서비스 게시와 발견 모델에 발견 에이전트(discovery agent)와 평판 관리자(reputation manager)를 추가하였다. 발견 에이전트는 사용자의 기능적, QoS, 평판 요구를 만족하는 서비스를 찾기 위해 서비스 사용자, UDDI 레지스트리와 평판 관리자 사이에서 중개자로서 행동한다. 평판 관리자는 사용자들로부터 서비스 등급을 수집하고 처리한다. 그리고 발견 에이전트에 의해 요청될 때 서비스 평판 점수를 제공한다. 그러나 Z. Xu의 연구에서는 현 UDDI 레지스트리에서 제공하지 않는 광고 QoS 정보 제공을 가정하였으며, 사용자가 평판 관리자에 제공하는 서비스 등급은 객관적인

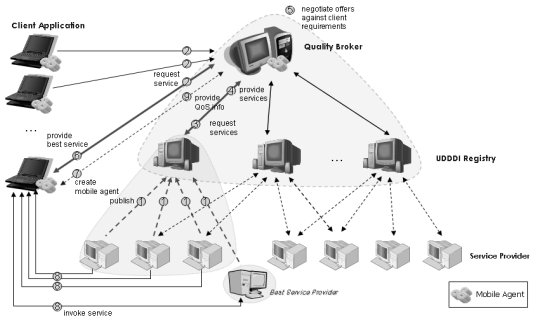
평가가 이루어져야함에도 불구하고 평가 방법을 설명하지 않았다. 즉 등급의 누락 또는 무효한 등급에 대한 처리는 다루지 않았다. 또한 이 연구에서 제안한 매칭, 랭킹, 선정 알고리즘에서는 사용자가 선택한 특정 품질 척도에 대해서만 QoS 점수를 계산하므로, QoS 점수 계산시 다른 품질 척도에 대한 고려는 이루어지지 않았다.

## 3. 품질 브로커 시스템 설계 및 구현

### 3.1 시스템 아키텍처 개요

본 절에서 제안하는 시스템 아키텍처는 (그림 1)과 같이 서비스 제공자, 사용자, UDDI 레지스트리로 구성되는 웹 서비스 아키텍처에 품질 브로커를 추가하였다.

품질 브로커는 애플리케이션(사용자)과 웹 서비스(제공자)를 상호 연결하는 역할을 한다. 브로커는 주어진 명세와 파라미터 등에 기초하여 적절한 웹 서비스를 찾아주는 "대리인"에 해당하는 프로그램이다. 즉, 품질 브로커는 최적의 QoS를 가진 서비스를 선정하고, 사용자들 사이에서 QoS에 관한 경험 정보를 공유하기 위해 사용된다. 다음 (그림 2)는 (그림 1)의 시스템 아키텍처에서 이루어지는 수행 프로세스를 단계별로 나타낸다.




(그림 1) 시스템 아키텍처

1. 서비스 제공자는 UDDI 레지스트리에 서비스를 공개한다.
2. 클라이언트 애플리케이션은 품질 브로커에 요청을 보내면서 요구되는 서비스 타입에 대한 정보와 요구되는 서비스 품질에 대한 정보를 함께 보낸다.
3. 품질 브로커는 가능성이 있는 웹 서비스를 찾기 위해 UDDI 레지스트리를 검색한다.
4. UDDI 레지스트리는 품질 브로커에 서비스들의 리스트를 제공한다.
5. 품질 브로커는 서비스들의 품질 정보와 클라이언트 애플리케이션의 품질 요구사항을 비교하여 가장 적합한 서비스를 선정한다.
6. 품질 브로커는 클라이언트 애플리케이션에 최적의 서비스에 관한 정보를 제공한다.
7. 품질 브로커는 이동 에이전트 객체를 원격으로 생성하고, QoS 정보의 모니터링을 수행하는 메시지를 에이전트 객체에 보낸다.
8. 클라이언트 애플리케이션은 서비스를 호출한다.
9. 이동 에이전트는 서비스의 사용동안 QoS 정보를 모니터링 한 뒤, 품질 브로커에 피드백 한다. 저장된 QoS 정보들은 향후 서비스 선정 프로세스에서 반복적으로 사용된다.

(그림 2) 제안 아키텍처의 수행 프로세스

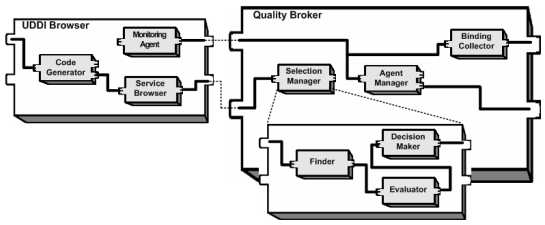
### 3.2 시스템 세부 구조

품질 브로커 시스템은 (그림 3)에서 보듯이 크게 UDDI 브라우저(UDDI Browser)와 품질 브로커(Quality Broker)로 구분된다.

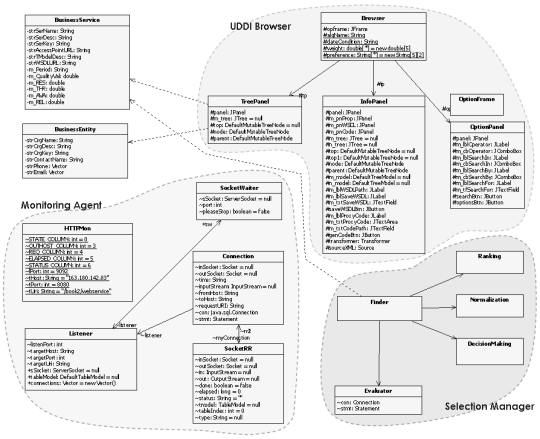
UDDI 브라우저는 사용자 위치에서 품질 브로커에 접속하여 Business, Service 정보를 검색한다. UDDI 브라우저는 검색된 객체의 결과를 보여주는 서비스 브라우저(Service Browser)와 선택된 서비스의 WSDL 파일로부터 프록시 클라이언트를 생성하는 코드 생성기(Code Generator)로 구성된다. 그 외에 모니터링 에이전트는 수동적인 방법을 이용해서 웹 서비스의 바인딩 정보를 수집한다.

품질 브로커는 크게 서비스 선정 관리자(Service Selection Manager), 에이전트 관리자(Agent Manager)로 구분된다. 서비스 선정 관리자는 사용자의 트랜잭션 정보에 근거해 최적의 서비스를 선정하고 그 결과를 사용자 측의 UDDI 브라우저에 제공한다. 서비스 선정 관리자는 다시 발견자(Finder), 평가자(Evaluator), 의사결정자(DecisionMaker)로 나누어진다. 발견자는 품질 척도 측정은 평가자에, 서비스 선정은 의사결정자에 의뢰한 후 서비스들을 순위대로 정렬한다. 평가자는 바인딩 DB에 저장된 서비스들의 바인딩 정보를 근거로 품질 척도 값들을 측정한다. 마지막으로 의사결정자는 평가자의 측정 결과를 토대로 최적의 서비스를 선정한다. 에이전트 관리자는 서비스 선정이 이루어지면 해당 서비스에 대한 모니터링 에이전트를 사용자 위치에서 원격 생성한다.

다음 (그림 4)는 제안 시스템을 구성하는 클래스와 각 클래스의 속성을 식별한 클래스 다이어그램을 나타내었다.




(그림 3) 품질 브로커 시스템 세부 구조




(그림 4) 클래스 다이어그램

### 3.3 품질 모니터링 에이전트

본 절에서는 전송 패킷의 프로토콜과 품질 모니터링 에이전트의 원격 생성 과정, 그리고 패킷 정보 수집 활동에 대하여 설명한다.

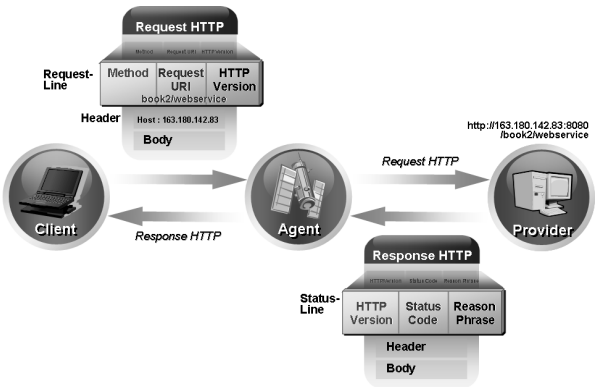
#### 3.3.1 바인딩 정보의 수집

서비스 사용자와 제공자 사이의 SOAP 메시지는 전송 프로토콜에 의해 배달이 되어야 한다. SOAP과 전송 프로토콜을 접목 시키는 작업을 바인딩(binding)이라고 한다. 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.

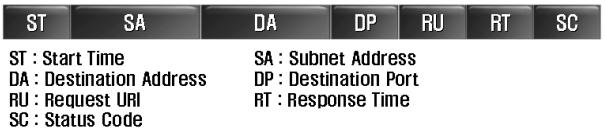
모니터링 에이전트는 (그림 5)와 같이 서비스 사용자에서 보내는 요청 HTTP 내용을 받아서 바인딩 정보를 수집 하고, 똑 같은 내용을 서비스 제공자 쪽으로 다시 보낸다. 마찬가지로 서비스 제공자 쪽에서 오는 응답 HTTP 내용에서 바인딩 정보를 수집하고 똑같은 내용을 서비스 사용자에게로 다시 보내 준다.

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[9]. 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.

모니터링 에이전트의 바인딩 정보 수집 후에는 에이전트와 브로커 서버 사이에 미리 약속한 일정한 형식인 프로토콜을 정의해야 한다. 모니터링 에이전트가 브로커 서버로 보내는 데이터 단위인 패킷 정보는 브로커 서버에서 해당 웹 서비스의 품질 척도를 측정하기 위해 필요로 하는 필드들로 구성된다. 패킷의 정확한 크기를 알 수 없기 때문에 패킷의 끝에는 '\'를 넣어 패킷의 끝임을 알려주고, 필드마다 '|'를 분리자(delimiter)로 넣어 필드를 구분하였다. 패킷을 받은 브로커 서버에서는 '\'가 나타날 때까지 하나씩 읽은 다음, 읽은 데이터를 ':'를 기준으로 끊어 토큰으로 분리한다. 전송 패킷의 프로토콜을 (그림 6)과 같이 정의하였으며 각 필드의 기능을 <표 1>에 정리하였다.




(그림 5) 모니터링 에이전트의 패킷 정보 수집



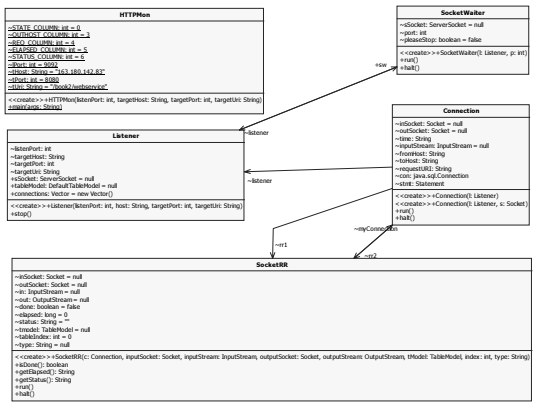

(그림 6) 전송 패킷의 프로토콜

〈표 1〉 패킷 정보의 필드 내용

| 필 드 명 | 획득 방법 | 기 능 |
|---|---|---|
| Start Time | Date() | 서비스 호출 시작 시간 |
| Subnet Address | | 클라이언트가 속한 서브넷 주소 |
| Destination Address | Request HTTP/Host | 제공자 측 주소 |
| Destination Port | | 제공자 측의 Tomcat 서버를 구분하는 포트번호 |
| Request URI | Request HTTP/Request-URI | 제공자 측에서 실행할 컴포넌트 |
| Response Time | EndTime-StartTime | 서비스 응답 시간 |
| Status Code | Response HTTP/Status Code | 클라이언트의 요청에 대한 성공 여부를 나타내는 코드<br>- 200(HTTP_OK) : 클라이언트 요청이 성공적으로 수행<br>- 500(HTTP_SERVER_ERROR) : 제공자에서 예기치 못한 상황으로 처리되지 못함<br>- 503(HTTP_UNAVAILABLE) : 제공자에서 일시적으로 요청을 처리하지 못함 |

#### 3.3.2 모니터링 에이전트의 구조

다음 (그림 6)은 모니터링 에이전트의 구조를 클래스 다이어그램으로 도식화한 것이다. 각 클래스의 개략적인 설명은 다음과 같다.



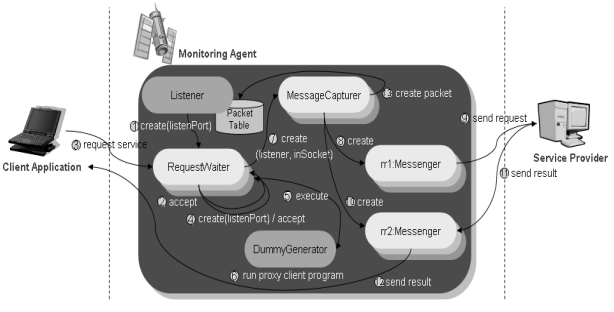

(그림 6) 에이전트의 클래스 다이어그램

HTTPMon 클래스는 listenPort, targetHost, targetPort, requestUri를 인자로 하는 Listener 객체를 생성한다. Listener 클래스는 바인딩 정보가 저장되는 패킷 테이블을 생성하고, listenPort를 인자로 하는 SocketWaiter 쓰레드를 생성한다. SocketWaiter 쓰레드에서는 listenPort를 통해 들어오는 연결마다 하나의 Connection 쓰레드를 생성한다. Connection 쓰레드에서는 요청 HTTP에서 요청시간, 클라이언트 호스트 주소, 요청 URI 정보를 추출 후, SocketRR 쓰레드 객체인 rr1, rr2를 통해 응답 시간과 상태 코드를 Listener 객체의 패킷 테이블에 저장한다. 마지막으로 SocketRR 쓰레드는 요청 또는 응답 HTTP의 전달을 담당한다.

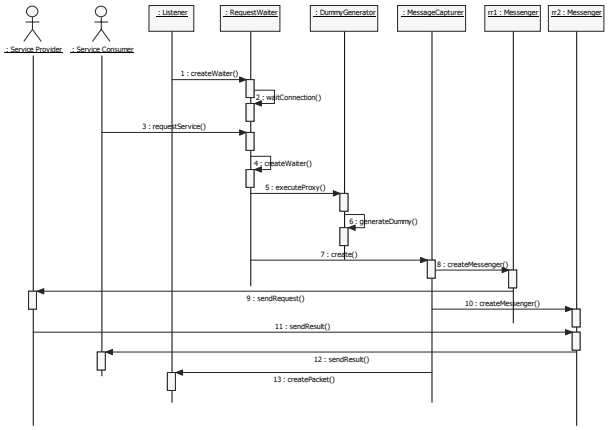
모니터링 에이전트는 서비스 클라이언트 측과 서비스 제공자 측이 서로 주고받는 SOAP요청/응답 메시지를 인터셉트(intercept)한 후, 해당 메시지를 전달하는 HTTP 구조에서 필요한 바인딩 정보를 획득한다. SOAP 요청/응답 메시지를 확인하기 위해서 먼저 클라이언트 소스코드 상의 endpoint URL의 포트가 listenPort로 변경되어야 한다. 이로서 요청/응답 메시지는 에이전트가 사용하는 listenPort로 전송되며, 에이전트는 바인딩 정보 수집의 본래 목적 외에도 SOAP 메시지 전달도 수행되어야 한다. 물론, 정상 운용 시에는 원래 포트로 복원한 후 이용해야 한다.

오직 선정된 서비스만을 대상으로 하는 피드백의 주요 문제는 기아 상태(starvation)이다. 낮은 순위의 서비스들이 무한하게 선정되지 못하는 기아 상태 문제에 대한 해결방안은 모니터링(monitoring)이다. 모니터링이란 이동 에이전트의 성능 모니터링을 선정된 서비스 외의 등록된 서비스들로 확대함으로서 결과적으로 순위를 증가시키는 기법이다. 예를 들어, 만약 선정 순위가 1에서 50까지의 범위라면, 한 명의 서비스 사용자에 의해 서비스가 선정될 때마다 선정 서비스뿐만 아니라 나머지 49개의 서비스도 성능을 측정한다. 결국에는 초기의 선정 순위가 50이었던 웹 서비스도 품질이 좋아진다면 상위 순위를 갖게 되어 선정 가능성이 높아진다.

다음 (그림 7)과 (그림 8), (그림 10)은 에이전트가 바인딩 과정의 모니터링을 통해 패킷 정보를 얻는 과정을 설명한 그림이다.




(그림 7) 모니터링 에이전트의 패킷 정보 수집

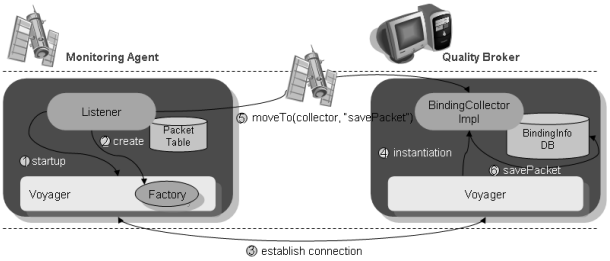



(그림 8) 에이전트의 패킷 정보 수집 순차 다이어그램

1. Listener 객체의 생성자는 검색 결과상의 서비스들의 주소, 포트번호, 실행 컴포넌트 정보로 구성된 배열을 매개변수로 받는다. Listener 객체는 listenPort(9090)를 인자로 SocktWaiter 객체를 쓰레드로 생성한다.
2. RequestWaiter 쓰레드는 포트로 들어오는 연결 요청을 기다린다.
3. 클라이언트 애플리케이션은 서비스를 요청한다.
4. 연결 시도에 RequestWaiter 쓰레드는 선정 서비스를 제외한 서비스 배열의 요소마다 하나의 RequestWaiter 쓰레드를 생성하고 연결 요청을 기다린다.
5. 등록서비스들에 대한 프록시 클라이언트 프로그램을 실행하는 DummyGenerator 객체를 호출한다.
6. DummyGenerator 객체는 UDDI 브라우저에 의해 생성된 프록시 클라이언트 프로그램들을 컴파일, 실행함으로서 더미 메시지를 발생시킨다.
7. 더미 메시지에 의해 RequestWaiter 쓰레드는 연결마다 하나의 MessageCapturer 쓰레드를 생성한다.
8. MessageCapturer 쓰레드에서는 요청 HTTP에서 요청시간, 클라이언트 호스트 주소, 요청 URI 정보를 추출 후, Messenger 쓰레드 객체인 rr1을 생성한 후, 실행한다.
9. rr1쓰레드에서는 요청 HTTP의 내용을 제공자의 서비스에 전달한다.
10. MessageCapturer 쓰레드에서는 Messenger 쓰레드 객체인 rr2를 생성한 후, 실행한다.
11. rr2쓰레드에서는 응답 HTTP에서 성공 여부 코드를 추출하고, 응답 시간을 계산한다.
12. rr2쓰레드에서는 선정된 서비스에 대한 응답 HTTP 내용만을 클라이언트 애플리케이션에 전달한다.
13. MessageCapturer 쓰레드에서는 모니터링 정보를 기반으로 Listener 객체의 tableModel에 패킷 정보를 작성한다.

(그림 9) 에이전트의 패킷 정보 수집 프로세스

각각의 사용자 위치에서 패킷 정보 수집이 끝나면, 모니터링 에이전트의 Packet Table에 저장된 바인딩 정보는 품질 브로커의 BindingInfo DB로 이동, 저장되어야 한다. 모니터링 에이전트에 이동성을 부여하는 이유는 서비스 선정 이후 클라이언트 프로그램의 서비스 요청이 언제 발생할지 모르기 때문이다. 즉 서비스 사용자 대부분은 최적의 서비스 선정을 확인한 후에 품질 브로커와의 접속을 끊고 클라이언트 프로그램을 통해 서비스 요




(그림 10) 모니터링 에이전트의 이동

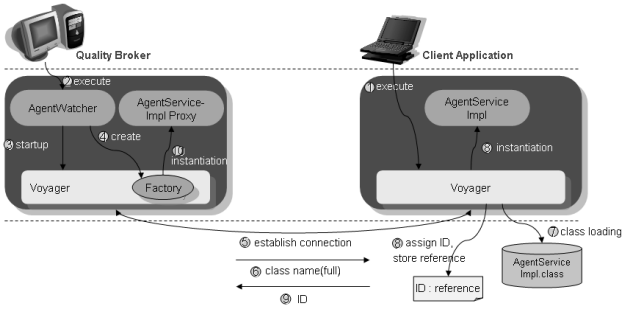
1. Voyager를 클라이언트로 실행시킨다.
2. Factory.create() 메소드를 통해 품질 브로커가 위치한 호스트 이름을 가지며, Voyager가 동작하는 품질 브로커에서 BindingCollector 서버 생성을 의뢰한다.
3. 품질 브로커 측 Voyager에 연결을 시도한다.
4. BindingCollector 구현 객체(collector)를 품질 브로커 호스트에서 생성한다.
5. 에이전트 객체는 collector 객체로 이동하여 savePacket()을 로컬하게 수행한다.
6. savePacket() 메소드는 BindingInfo DB에 에이전트의 Packet Table에 저장된 패킷 정보를 저장한다.

(그림 11) 에이전트의 이동 프로세스

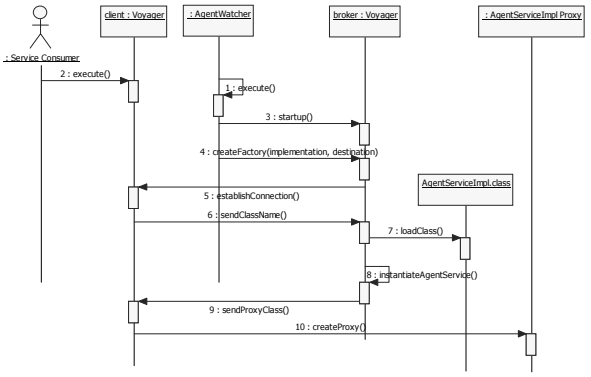
청을 하기 때문이다. 에이전트 객체는 객체 자체의 정보와 그 객체에서 유지되는 패킷 데이터가 직렬화 메커니즘을 통해 새로 지정된 품질 브로커 위치로 복사된다. 복사가 끝나면 원래의 에이전트 개체는 사용자측 호스트에서 없어진다. 다음 (그림 10)과 (그림 11)은 에이전트의 이동 과정을 이해하기 쉽게 설명한 그림이다.

### 3.4 모니터링 에이전트 관리자

다음 (그림 12)와 (그림 13), (그림 14)는 모니터링 에이전트의 원격 생성 과정을 이해하기 쉽게 설명한 그림이다. 모니터링 에이전트가 클라이언트측 호스트에서 원격 생성되면, 에이전트는 클라이언트 애플리케이션과 서비스 제공자 사이에서 주고받는 HTTP 메시지를 모니터링 한다.




(그림 12) 모니터링 에이전트의 원격 생성




(그림 13) 에이전트의 원격 생성 순차 다이어그램

1. Voyager 서버를 8000번 포트에서 실행시킨다.
2. 품질 브로커는 AgentWatcher를 실행한다.
3. AgentWatcher는 Voyager를 클라이언트로 실행시킨다.
4. Factory.create(String implementation, String destination)를 실행한다. implementation에는 생성할 객체의 타입을, destination에는 원격 객체를 생성하기를 원하는 프로그램의 주소를 지정해 준다.
5. 클라이언트 측 Voyager에 연결을 시도한다.
6. 클라이언트 측 Voyager에 원격 객체의 클래스 명을 전달한다.
7. Voyager의 네트워크 클래스 로더는 자동으로 AgentService 클래스 코드를 로딩 한다.
8. AgentService 구현 객체를 클라이언트 호스트에서 생성한다.
9. 새로 생성된 객체에 대한 프록시(proxy) 클래스를 생성한다. 프록시 클래스는 품질 브로커로부터 메시지를 받아 원격 객체에 전달한다. 반대로 원격 객체로부터 결과 값을 받아 품질 브로커에게 전달한다.
10. 프록시 클래스를 품질 브로커에 반환한다.

(그림 14) 에이전트의 원격 생성 프로세스

### 3.5 서비스 선정 관리자

#### 3.5.1 품질 척도의 측정

모니터링 에이전트로부터 수집 된 바인딩 정보를 토대로 해당 웹 서비스 품질의 척도들을 측정할 수 있다. 서비스별로 여러 건의 바인딩 정보가 저장되므로, 품질 척도를 측정하기 위해서는 사용자의 품질 척도 요구 시점을 기준으로 단위 시간(예, 일별, 주별, 월별, 년별) 동안의 평균값을 계산해야 한다. 따라서 각 바인딩 정보의 종료 시간을 근거로 단위 시간에 속하는지를 알 수 있다. 품질 척도들의 측정 방법은 다음과 같다.

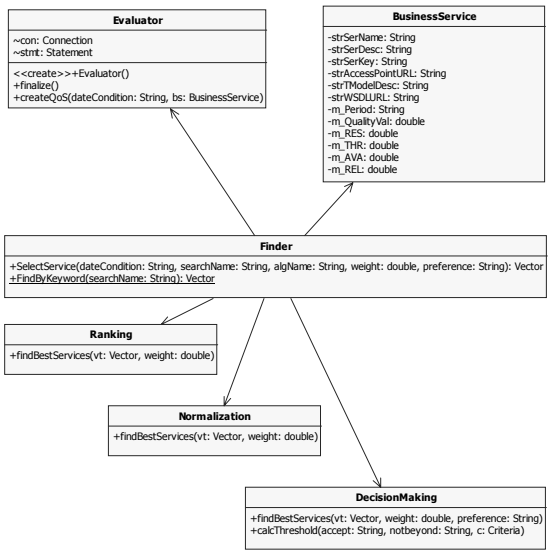
- 응답 시간을 측정하기 위해서 바인딩 정보의 상태 코드 값이 200이고 서브넷 주소가 서비스를 요청한 클라이언트가 속한 서브넷 주소와 일치하는 경우에 한해 평균 응답 시간으로 측정한다. 응답 시간과 같이 클라이언트가 속한 네트워크 환경에 따라 그 차이가 큰 품질 척도들의 신뢰성을 높이기 위해서는 같은 서

브넷에 속한 바인딩 정보로 국한한다.

- 처리량을 측정하기 위해서 단위 시간별로 사용자 요청이 성공적으로 수행된 경우에 서비스 호출 요청(바인딩 정보의 개수)의 비율로 측정한다.
- 가용성을 측정하기 위해서 바인딩 정보의 상태 코드 값을 사용한다. 단위 시간별로 전체 호출 건수(바인딩 정보의 개수)에 대하여 성공적인 실행 건수(코드 값 200)를 측정할 수 있다.
- 신뢰성을 측정하기 위해서 가용성과 마찬가지로 바인딩 정보의 상태 코드 값을 사용한다. 신뢰성은 단위 시간 동안의 고장 횟수(코드 값 500, 503)로 측정할 수 있다.

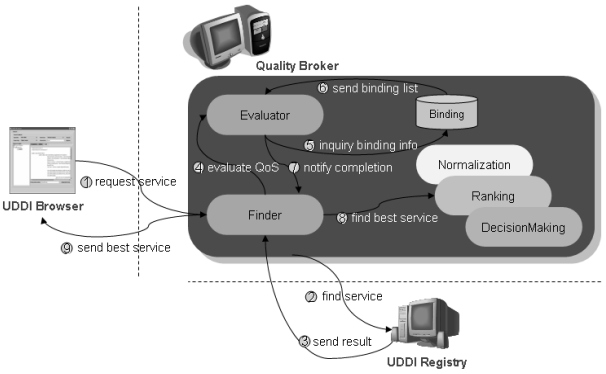
3.5.2 서비스 선정 관리자의 구조

서비스 선정 관리자는 키워드 검색을 통해 찾은 서비스들을 대상으로 품질 척도 값을 계산하고 사용자가 선택한 서비스 선정 알고리즘에 의뢰해 최적의 서비스를 통보 받는다. 다음 (그림 15)는 서비스 선정 관리자의 구조를 클래스 다이어그램으로 도식화한 것이다. 각 클래스의 개략적인 설명은 다음과 같다. Finder 클래스는 UDDI 레지스트리에서 키워드 검색을 통해 사용자의 기능적 요구를 만족하는 서비스들을 찾는다. 또한 품질 척도 값 생성을 Evaluator 클래스에 의뢰하고, 선택한 알고리즘을 통해 최적의 서비스를 선정한다. Evaluator 클래스에서는 Binding DB에 연결 후 서비스들마다 품질 척도 값을 계산한다. Ranking, DecisionMaking, Normalization 클래스는 서비스 선정 알고리즘을 구현한 클래스들이다. 마지막으로 BusinessService 클래스는 키워드 검색을 통해 찾은 서비스 객체들의 타입으로 사용된다.

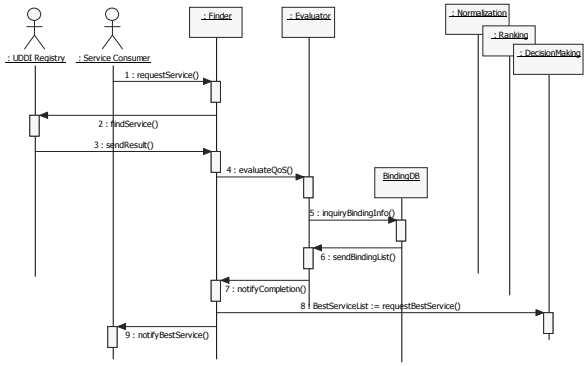



(그림 15) 선정 관리자의 클래스 다이어그램

다음 (그림 16)과 (그림 17), (그림 18)은 선정 관리자의 서비스 선정 과정을 이해하기 쉽게 설명한 그림이다.




(그림 16) 선정 관리자의 서비스 선정 과정




(그림 17) 선정 관리자의 서비스 선정 순차 다이어그램

1. UDDI 브라우저는 선정 관리자에 서비스 검색을 요청한다. 단, 요청시에는 기능적 검색어, 조회 기간, 선호 알고리즘, 품질 가중치가 필요하다.
2. 선정 관리자의 Finder 객체에서는 검색어와 일치하는 서비스들을 UDDI 레지스트리에서 찾는다.
3. UDDI 레지스트리는 일치하는 서비스들의 리스트를 Finder 객체에 보낸다.
4. 서비스 선정 알고리즘이 선택한 경우에 한해 Evaluator 객체에 서비스들의 품질 척도 평가를 의뢰한다.
5. Evaluator 객체는 Binding DB에 연결하고 해당 서비스에 일치하는 바인딩 정보를 질의한다.
6. Binding DB는 그 결과를 리턴하며, 5, 6의 과정은 서비스 리스트 상의 모든 서비스가 품질 척도 값을 구할 때까지 반복된다.
7. Finder 객체에 품질 척도 평가의 완료를 통보한다.
8. Finder 객체는 품질 척도 값에 의한 최적의 서비스를 선호 알고리즘 객체에 의뢰한다.
9. Finder 객체는 UDDI 브라우저에 최적의 서비스를 알려준다.

(그림 18) 선정 관리자의 서비스 선정 수행 프로세스

### 3.6 UDDI 브라우저

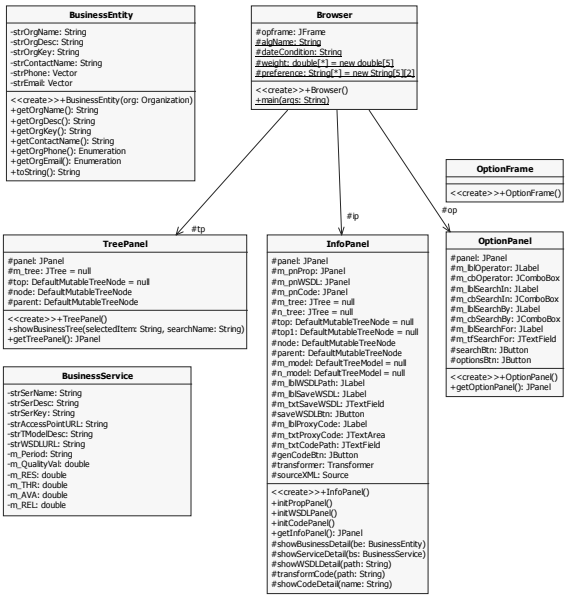
#### 3.6.1 UDDI 브라우저의 구조

다음 (그림 19)는 UDDI 브라우저의 구조를 클래스 다이어그램으로 도식화한 것이다.

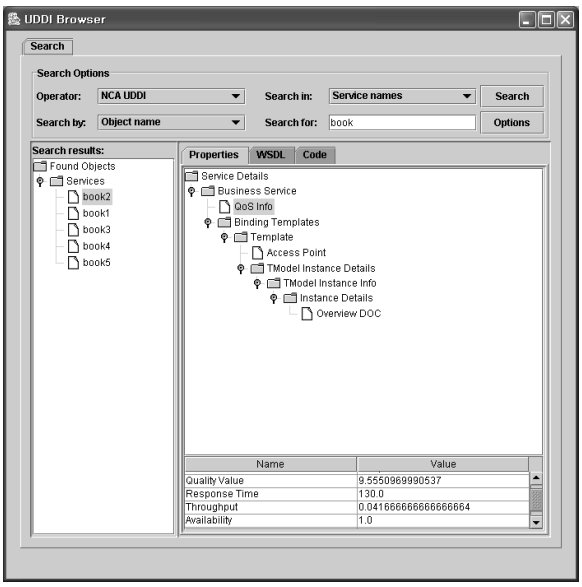
UDDI Browser는 질의어와 필터를 이용하여 Business와 Services, tModels와 같은 UDDI 객체들을 검색한다.

검색 결과는 Tree View, Properties 탭, WSDL 탭에 나타난다. Tree View에는 Search Options에서 질의어와 필터에 의해 검색된 객체의 결과를 보여준다. Properties 탭에는 Tree View에서 선택된 비즈니스와 서비스 결과의 항목을 각각 (그림 20)과 같이 객체 구조로 펼쳐 보여준다. 객체 구조에서 필드를 선택하면 해당 필드의 세부 정보가 밑의 박스에 리스트로 나타난다.

(그림 20)의 객체 구조에서 QoS Info 필드의 세부 정보에는 각 서비스마다 응답 시간, 처리율, 가용성, 신뢰성과 같은 품질 척도와 선정 알고리즘을 통해 계산된 품질 값이 나타난다.




(그림 19) UDDI 브라우저의 클래스 다이어그램




(그림 20) 서비스 검색 결과

## 4. 비교 평가

### 4.1 시뮬레이션

이번 시뮬레이션은 성능 모니터링을 선정된 서비스 외의 등록된 서비스들로 확대하는 모니터링 기법을 사용하는 아키텍처는 서비스의 기아 상태를 해결 수 있다는 것을 보여준다.

**서비스** : 9개의 서비스가 시뮬레이션을 위해 사용된다. 모든 서비스는 기능적으로 동일한 온라인 서점 웹 서비스[14]이며, 서비스들 사이의 차이점은 실제 QoS 성능이다. QoS 성능에 따른 순위에 의해 서비스들은 사용자들로부터 서로 다른 선택을 받는다.

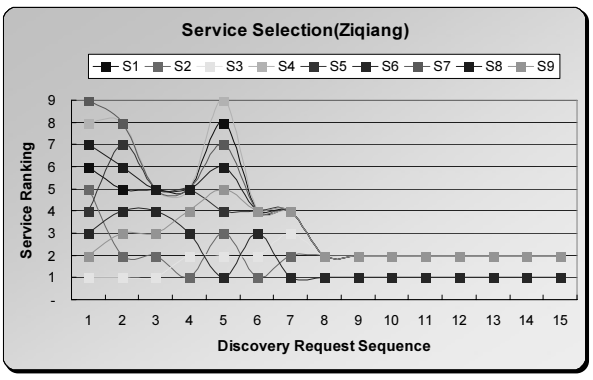
**사용자** : 사용자들은 <표 2>와 같이 서로 다른 수준의 QoS 요구를 가진다. QoS 품질 척도에 대한 가중치는 모두 0.2이며, 모든 사용자는 리턴 되는 서비스 들 중에서 요구하는 서비스 수준을 만족하는 서비스만을 선정한다고 가정한다.

**실험 결과** : 시뮬레이션은 각 사용자에 대해 동일한 서비스 발견 요청이 15번 수행된다.

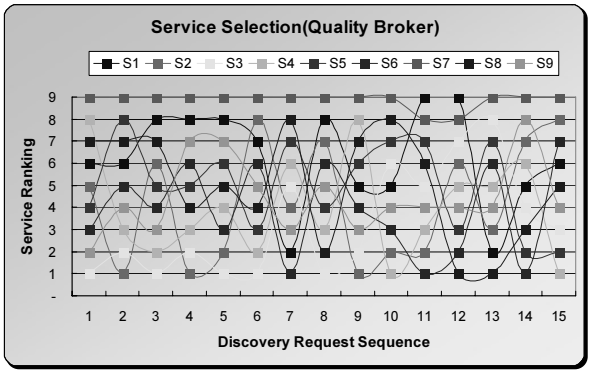
다음 (그림 21)은 앞서 관련 연구에서 살펴본 Ziqiang[3]이 제안한 아키텍처를 통해 서비스를 선정한 결과이다. 모니터링 기법을 사용하였지만 오직 선정된 서비스들에 대한 성능 정보만을 피드백 하기 때문에 서비스 발견 요

〈표 2〉 사용자들의 QoS 요구 정보

| QoS Requirements | | | |
|---|---|---|---|
| Response Time | Throughput | Availability | Reliability |
| gold | bronze | gold | gold |




(그림 21) 기존 아키텍처의 서비스 선정 결과




(그림 22) 제안 아키텍처의 서비스 선정 결과

청이 진행될수록 사용자의 QoS 요구를 만족하는 서비스만이 피드백 된다. 따라서 선정되지 못한 서비스들은 해당 서비스의 성능 정보가 피드백 되지 않으므로, 설사 성능이 향상되더라도 기아 상태가 발생한다.

다음 (그림 22)는 제안 아키텍처인 품질 브로커를 통해 서비스를 선정한 결과이다. 최적의 서비스를 선정하기 위해 다기준 의사 결정 기법중[15] 하나인 PROMETHEE를 이용한 알고리즘을 사용하였다[16][17]. 품질 브로커에서는 선정된 서비스 외의 등록된 서비스들에 대한 피드백을 통해 기아 상태를 해결하였다. 즉 등록된 서비스들은 사용자의 품질 요구에 만족하지 못하여 선정되지 못하더라도 계속 해당 서비스의 성능 정보가 피드백 되기 때문에 향후 요구에 만족할 수준의 성능 향상이 있다면 선정될 수 있다.

### 4.2 비교 평가

앞서 관련 연구에서 기술한 기존 연구들과 본 논문에서 제안한 연구와의 비교 평가는 <표 3>과 같으며, 각각의 특징을 살펴보면 다음과 같다.

첫째, 선정 메커니즘의 사용 시점. Moor의 연구[10]를 제외하고는 비교 대상이 되는 모든 연구들은 실행 시간에 동적으로 웹 서비스를 선정하였다. 이에 반해 Moor의 연구는 사용자와 커뮤니티 멤버들이 설계 시간에 선정 메커니즘을 수행하였다.

둘째, 선정 프로세스에 필요한 정보의 종류. Moor의 연구를 제외하고는 대부분의 연구들은 기본적으로 QoS 데이터를 선정 프로세스에서 사용하였다. Ran[6], Yu[11], Xu[3]의 연구에서 QoS 데이터는 서비스 제공자가 제공하는 광고 QoS 데이터이며, 신뢰성 문제를 해결하기 인해 Ran의 연구에서는 증명자를 통해 광고 QoS를 보증하는 과정을 추가하였다. 그 외 Yu와 Xu의 연구에서는 광고 QoS와 함께 서비스 사용자의 평판 정보를 사용하였다. 본 논문에서 제안한 연구에서는 Liu[12], Day[13], Max[1]의 연구와 같이 광고 QoS를 고려하지 않았다. 광고 QoS는 신뢰성을 보증할 수 없는 문제점이 있다. 서비스 제공자가 더 많은 사용자들을 끌어들이기 위해 부정확한 QoS 정보를 게시하거나, 게시된 QoS 정보가 오래되었을 수도 있다. 따라서 본 논문에서는 현 UDDI 레지스트리에서 제공하지 않는 광고 QoS 정보사용을 배제하였으며, 서비스 사용자들이 서비스를 사용하는 시점에서 겪는 성능, 안전성, 비용 정보에 기반한 QoS 데이터를 사용하였다.

셋째, 정보의 획득 방법. Liu와 Day의 연구에서는 QoS 데이터를, Yu와 Xu의 연구에서는 평판 정보를 클라이언트가 직접 QoS 레지스트리 또는 포럼 시스템에 피드백 한다. 이러한 연구들의 문제점은 서비스 사용자들이 서비스와의 상호작용 이후에 QoS 데이터나 평판 정보를 피드백 하는 절차를 강제할 수 없다는 점이다. 따라서 이러한 사용자에 자율적인 피드백은 관련 정보의 누락 가능성과 신뢰성 문제를 야기할 수 있다. 그에 반해 Max의 연구는 서비스마다 에이전트를 사용해 QoS 데이터를 가지고 오는 과정에서 누락 가능성 문제는 피할 수 있으나, 사용자가 체험한 QoS 데이터가 아니라는 점에서 신뢰성 문제를 야기할 수 있다. 제안 연구에서는 서비스 사용자가 서비스를 사용하면서 체험하는 성능 정보를 에이전트의 모니터링을 통해 브로커에 피드백 함으로서 사용자의 개입을 최소화 한다. 즉 사용자를 대신한 자율적인 정보 피드백이 가능하다.

넷째, 기아상태의 해결. Liu와 Day의 연구에서와 같이 선정 프로세스에서 광고 QoS를 사용하지 않는 연구에서

〈표 3〉 서비스 선정 연구의 비교 평가

| 연구 / 특징 | Moor [10] | Ran [6] | Liu [12] | Yu [11] | Day [13] | Xu [3] | Max [1] | 제안 연구 |
|---|---|---|---|---|---|---|---|---|
| 사용시점 | 설계시점 | 실행시점 | 실행시점 | 실행시점 | 실행시점 | 실행시점 | 실행시점 | 실행시점 |
| 정보종류 | 사용자, 커뮤니티 요구사항 | 보증된 광고 QoS | QoS 데이터 | 광고 QoS, 평판 정보 | QoS 데이터 | 광고 QoS, 평판 정보 | QoS 데이터 | QoS 데이터 |
| 획득방법 | 개발 주기에서 사용자들을 포함 | 서비스 제공자의 등록 | 클라이언트의 피드백 | 서비스 제공자의 등록, 클라이언트의 피드백 | 클라이언트의 피드백 | 서비스 제공자의 등록, 클라이언트의 피드백 | 에이전트의 피드백 | 에이전트의 피드백 |
| 기아상태 | 해당사항 없음 | 해당사항 없음 | 기아상태 발생 | 해당사항 없음 | 기아상태 발생 | 해당사항 없음 | 해당사항 없음 | 등록서비스들에 대한 에이전트의 모니터링으로 해결 |

는 기아상태(Starvation)가 발생한다. 즉 신규 서비스들은 높은 성능을 제공하더라도 품질 정보가 없기 때문에 선정 순위에서 계속적으로 낮은 순위를 받게 된다. 그에 반해 제안 연구에서는 선정된 서비스 외에도 신규 서비스를 포함한 등록된 서비스들에 대한 품질 정보를 이동 에이전트의 모니터링을 통해 수집하여 피드백 함으로서 기아 상태를 해결할 수 있다.

## 5. 결론과 향후 연구과제

본 논문에서는 서비스 사용자가 요구하는 기능과 QoS를 만족하는 서비스를 자율적으로 선정하는 품질 브로커 아키텍처를 제안하였다. 본 논문에서 제안한 선정 아키텍처는 전통적인 서비스 발견 모델의 테두리를 벗어나지 않고 실행 시간에 동적인 웹 서비스 발견을 위한 해결 방안을 제공한다. 또한 제안된 아키텍처에서 선정 관리자는 에이전트에 의해 수집된 품질 정보를 바탕으로 서비스 선정 문제를 해결함으로서 클라이언트 애플리케이션에 최적의 서비스를 제시하였다. 에이전트는 사용자가 서비스를 사용하면서 인식하는 성능을 모니터링 하기 위해 사용자 위치에서 생성된다. 또한 서비스 선정 이후 클라이언트 프로그램의 서비스 요청이 언제 발생할지 모르기 때문에 본 논문에서는 에이전트에 이동성을 부여하였다. 서비스 사용자 대부분은 최적의 서비스 선정을 확인한 후에 품질 브로커와의 접속을 끊고 클라이언트 프로그램을 통해 서비스 요청을 하기 때문에 이를 해결하기 위해서는 자율적으로 동작하는 에이전트가 필요하였다.

향후 연구로는 에이전트의 기능을 바인딩 정보 수집 외에 품질 정보 감시로 확대할 필요가 있다. 에이전트는 실행시간에 선정한 서비스의 QoS 값 변화를 계속적으로 측정하면서 일정 시간 사용자의 품질 기대치 이하로 QoS 값이 감소한다면 사용자에 통보하는 능력이 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] M. Ouzzani and A. Bouguettaya, “Efficient Access to Web Services,” IEEE Internet Computing, Vol.8, No.2, pp.34-44, Mar., 2004.

[3] Z. Xu, “Reputation-Enhanced Web Services Discovery with QoS,” M.S. Thesis, School of Computing, Queen's University, Canada, Aug., 2006.

[4] Tom Bellwood, Luc Clément, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Januszewski, Sam Lee, Barbara McKee, Joel Munter, Claus von Riegen, “UDDI Version 3.0 Published Specification,” July, 2002.(See http://uddi.org/pubs/uddi_v3.htm)

[5] 한국 썬 마이크로시스템즈, “UDDI 레지스트리의 가능성 탐구,” Sun microsystems monthly newsletter, pp. 18-25, 2003.

[6] S. Ran, “A model for web services discovery with QoS,” ACM SIGecom Exchanges, Vol.4, Issue.1, pp.1-10, Spring. 2003.

[7] A. Shaikhali, et al., “UDDIe: An extended registry for Web Services,” Symposium on Applications and the Internet Workshops(SAINT '03 Workshops), 2003.

[8] P. Farkas and H. Charaf, “Web Services Planning Concepts,” 1st International Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Distributed and WEB Computing, Feb., 2003.

[9] R. Fielding, et al., “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2068, Jan. 1997.

[10] A.D. Moor and W.J.V.D. Heuvel, “Web service selection in virtual communities,” In 37thHawaii International Conference on System Sciences, Jan., 2004.

[11] Tao Yu and K.J Lin, “Service Selection Algorithms for Web Services with End-to-end QoS Constraints,” Proceedings of the IEEE International Conference of E-Commerce Technology, Vol.00, pp.129-136, 2004.

[12] Y. Liu, A. Ngu and L. Zheng, “QoS computation and policing in dynamic web service selection (to appear)” In Proceedings of the WWW 2004, May, 2004.

[13] J. Day and R. Deters, “Selecting the Best Web Service,” In Proceedings of the 14th Annual IBM Centers for Advanced Studies Conference(CASCON), pp.293-307, Oct., 2004.

[14] 신민철, “기초에서 실무까지 XML 웹 서비스,” (주)프리렉, 2004.

[15] 박상현, 양성봉, “다중 속성 협상과 상호 이익을 위한 중개 에이전트 시스템,” 한국정보과학회 논문지, 31권, 3호, pp.308-316, Mar, 2004.

[16] Y.J. Seo, H.Y. Jeong and Y.J. Song, “Best Web Service Selection Based on the Decision Making Between QoS Criteria of Service,” LNCS Vol.3820, pp.408-419, Springer-Verlag, 2005.

[17] P.N. Kodikara, B.J.C. Perera and M.D.U.P. Kularathna, “Stakeholder Preference Modelling In Multi-Objective Operation Of Urban Water Supply Systems-A Case Study On Melbourne Water Supply System,” International Congress on Modelling and Simulation, pp.1539-1545, 2005.


## 서 영 준

e-mail : yjseo@khu.ac.kr

1999년 경희대학교 전자계산공학과 (공학사)

2001년 경희대학교 대학원 전자계산공학과(공학석사)

2007년 경희대학교 대학원 전자계산공학과(공학박사)

2007년~현 재 국가기록원 대통령기록관 공업연구사

관심분야 : 웹 서비스, CBSE, 소프트웨어 재사용 등



## 송 영 재

e-mail : yjsong@khu.ac.kr

1969년 인하대학교 전기공학과(공학사)

1976년 일본 Keio University 대학원 전산학과(공학석사)

1979년 명지대학교 대학원 전산학과 (공학박사)

1984년~1989년 경희대학교 전자계산소장

1993년~1995년 경희대학교 교무처장

1996년~1998년 경희대학교 공과대학장

1998년~2000년 경희대학교 기획조정실장

2001년~2002년 경희대학교 산업정보대학원장

1976년~현 재 경희대학교 전자정보학부 교수

관심분야 : 웹 서비스, CBSE, CASE 도구, 소프트웨어 재사용 등