SOA 서비스의 동적 선택 설계 기법
(A Design Method for Dynamic Selection of SOA Services)

배 정 섭 † 라 현 정 † 김 수 동 ††
(Jeong Seop Bae) (Hyun Jung La) (Soo Dong Kim)

요 약 서비스 지향 컴퓨팅(Service-Oriented Computing, SOC)은 배포된 서비스를 선택하고 조합하여 서비스 클라이언트가 원하는 기능을 제공하는 개발 방식이다. SOC는 향상된 비즈니스 기민성, 단축된 개발 시간과 같은 여러 장점을 제공한다. 이러한 장점을 극대화하기 위해서는 서비스의 선택과 조합이 동적으로 이루어져야 한다. 하지만 현재의 프로그래밍 언어, SOC 플랫폼, 비즈니스 프로세스 모델링 언어(Business Process Modeling Language, BPML) 및 도구는 수동적 서비스 선택 또는 서비스의 정적 바인딩만을 지원하는 수준에 머물러 있다. 각 클라이언트의 요구사항을 만족하는 서비스를 제공하기 위하여 해당 비즈니스 프로세스는 재구성 (reconfiguration)되고 재배포 (redeploy)되어야 하는 문제점이 있다. 따라서, 서비스 클라이언트의 다양한 요구에 맞게 서비스를 신속하고 유연하게 조합시키기 위하여 동적 선택 기법이 필요하다. 본 논문에서는 엔터프라이즈 서비스 버스(Enterprise Service Bus, ESB) 기반의 동적 선택 핸들러 (Dynamic Selection Handler, DSH) 설계 기법을 제안한다. DSH의 네 가지 컴포넌트인 수행 리스너, 서비스 선택자, 서비스 바인더, 인터페이스 변환자에 대한 설계를 제시한다. DSH 설계 시에 적합한 디자인 패턴을 적용하여 컴포넌트의 재사용성이 높도록 설계한다. 마지막으로 제안한 DSH 설계의 실용성을 보이기 위해 ESB를 이용하여 DSH를 구현한다.

키워드 : 서비스 지향 아키텍처(SOA), 동적 서비스 선택, Enterprise Service Bus (ESB)

Abstract Service-Oriented Computing (SOC) is the development method that published services are selected and composed at runtime to deliver the expected functionality to service clients. SOC should get maximum benefits not only supporting business agility but also reducing the development time. Services are selected and composed at runtime to improve the benefits. However, current programming language, SOC platforms, business process modeling language, and tools support either manual selection or static binding of published services. There is a limitation on reconfiguring and redeploying the business process to deliver the expected services to each client. Therefore, dynamic selection is needed for composing appropriate services to service clients in a quick and flexible manner. In this paper, we propose Dynamic Selection Handler (DSH) on ESB. we present a design method of Dynamic Selection Handler which consists of four components; Invocation Listener, Service Selector, Service Binder and Interface Transformer. We apply appropriate design patterns for each component to maximize reusability of components. Finally, we describe a case study that shows the feasibility of DSH on ESB.

Key words : Service Oriented Architecture (SOA), Dynamic Service Selection, Enterprise Service Bus (ESB)

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 학생회원 : 숭실대학교 컴퓨터학과
jsbae@otlab.ssu.ac.kr
hjla@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr

논문접수 : 2007년 9월 11일
심사완료 : 2008년 1월 4일

정보과학회논문지: 소프트웨어 및 응용 제35권 제2호(2008.2)

1. 서 론

서비스 지향 컴퓨팅(Service-Oriented Computing, SOC)은 다양한 서비스 클라이언트에게 서비스를 제공하기 위하여 적합한 형태로 여러 서비스를 적응(adaptation)하여 실행 시간에 선택 한 후 조합(composition)한다. 이와 같이 작업들을 "동적(dynamic)"으로 수행하는 것이 SOC의 특징 중 하나이다. 동적 서비스 선택은 서비스 클라이언트 프로그램을 변경시키지 않고 실행 시간에 배포된 서비스를 발견하여 클라이언트의 요구에 맞는 서비스 조합을 가능하게 한다. 동적 서비스 선택 기법을 이용하면, 서비스 클라이언트의 요구에 맞는 서비스를 선택할 수 있을 뿐만 아니라 클라이언트가 서비스를 호출할 때 컨텍스트에 맞는 서비스를 선택할 수 있다. 그러므로, 한정된 서비스를 이용하여 서비스 클라이언트의 다양한 요구에 맞는 서비스를 유연하게 조합시킬 수 있고, 실행 시간에 서비스를 적응시킬 수 있다. 그러나 현재의 프로그래밍 언어, SOC 플랫폼, 비즈니스 프로세스 모델링 언어(Business Process Modeling Language, BPML), 및 도구는 동적 서비스 선택을 충분히 지원하지 못하고 있다. 현재의 기술은 동적 서비스 선택 보다는 설계 시점에서 서비스를 결정하고, 적합한 서비스를 바인딩 하는 정적 조합을 지원하는데 초점이 맞추어져 있다.

본 논문에서는 많이 사용되고 있는 버스 아키텍처인 Enterprise Service Bus(ESB)을 이용하여 동적 서비스 선택을 가능하게 하는 실용적인 설계 기법을 제안한다. 2장에서 동적 서비스 선택과 관련된 기존 연구와 ESB에 대해 간략히 요약하며, 3장에서 기존에 수행된 연구에 대한 분석 결과를 기반으로 동적 서비스 선택을 접근하는 방법과 본 논문의 동적 서비스 선택의 범위에 대해 알아본다. 4장에서 동적 서비스 선택을 위한 설계 기법인 Invocation Listener, Service Selector, Service Binder, Interface Transformer로 이루어진 동적 선택 핸들러(Dynamic Selection Handler, DSH) 설계 기법을 제안한다. 이 DSH을 이용하여, 서비스는 동적으로 발견되고, 클라이언트가 원하는 요구에 맞는 서비스를 선택하고 목적에 맞게 적응시킬 수 있다. 마지막으로, 5장에서 제안된 기법의 실행 가능성을 보여주기 위해 호텔 예약 서비스를 대상으로 사례 연구를 수행한다.

2. 연구 배경

2.1 서비스 선택에 관한 관련 연구

Fujii는 Component Service Model with Semantics (CoSMoS), Component Runtime Environment(CoRE), Semantic Graph-Based Service Composition(SeGSeC)의 주요 컴포넌트로 구성된 시맨틱 기반의 동적 서비스 컴포지션 시스템을 제안하였다[1]. 이 주요 컴포넌트들은 시맨틱 정보를 인식하고 분석하며, 이 정보를 기반으로 서비스를 조합할 수 있는 기능을 수행한다. CoSMoS는 오퍼레이션과 속성으로 명세 된 컴포넌트이다. CoSMoS는 개념 (concept)이라는 용어를 소개했다. 개념을 이용하여 추상적인 지식 (idea)과 활동 (action)을 나타내는 엔티디와 컴포넌트의 오퍼레이션, 입력, 출력, 속성의 시맨틱을 나타낸다. CoRE는 다양한 컴포넌트 기술을 CoSMoS가 제공하기 위한 미들웨어이다. SeGSeC는 클라이언트가 요청한 시맨틱 정보를 기반으로 여러 후보 서비스 컴포넌트로부터 특정 서비스 컴포넌트를 조합하는 메커니즘을 제공한다. 서비스 조합 메커니즘은 시맨틱 그래프를 이용하여 클라이언트 요청을 생성하고 Service Composer는 요청에 맞는 워크플로우를 생성한다. Semantics Analyzer는 시맨틱 매칭을 수행하고 Service Performer는 워크프로우를 수행한다.

Penta는 기능적 비기능적 선호도와 제약사항에 따른 서비스 조합을 동적으로 바인딩 하기 위한 프레임워크인 WS Binder를 제안하였다[2]. 이 프레임워크는 두 가지 종류의 바인딩인 수행 전 바인딩, 실행 시간 바인딩을 제공한다. 수행 전 바인딩은 수행 전 시간에 가용한 서비스를 발견하여 조합하기 위한 것이고 실행 시간 바인딩은 조합 시간에 프락시가 실제 서비스를 수행한다. 실행 시간 바인딩은 가용하지 않는 서비스를 위하여 서비스 발견과 조합이 수행된다. 이러한 바인딩들은 QoS에 기반하여 워크플로우 제약사항을 정의한다. 워크플로우 내의 추상 서비스와 관련된 선호도를 고정한 후 서비스들 간의 의존성을 식별하고 재 바인딩과 관련된 선호도를 정의한다. 동적으로 비즈니스 프로세스에서 정의된 추상 서비스를 수행하기 위하여 프락시를 이용하여 적합한 서비스를 수행한다.

Yu는 QoS 기반 서비스 선택을 쉽게 하기 위하여 브로커 기반 아키텍처를 설계하였다[3]. 서비스 선택의 목표는 QoS 요구사항에 부합하는 서비스를 조합하고 QoS 속성을 최대화 또는 최소화 하여 어플리케이션의 활용 기능성을 극대화 하는 것이다. 순차적 흐름 구조에서 서비스 선택 알고리즘을 설명한다. 서비스 선택에 관한 문제를 관련된 0-1 multidimension multichoice knapsack problem(MMKP) 알고리즘을 설명한다. 문제가 커지면 커질수록 수행 시간이 기아급수적으로 증가하는 문제점을 해결하기 위하여 WS_HEU 알고리즘을 사용한다. 그래픽 모델에서 서비스 선택 문제를 해결하기 위하여 제약 요구사항을 해결하기 위하여 MCSP 알고리즘을 제안한다. 실 세계의 서비스 프로세스는 순차적 흐름 구조와 같이 고정되어 있지 않기 때문에 일반적인 흐름

구조에 관한 서비스 선택 알고리즘을 설명한다. 조합 모델을 이용하여 일반적인 흐름 구조에서 서비스를 선택하기 위하여 선택에 관한 문제를 0-1 Integer Programming(0-1 IP)로 고려하여 WS_IP 알고리즘을 디자인하였다. IP 문제에 있어서 최적의 결과를 찾기 위한 최악 경우 수행시간(worst-case computation time)은 문제의 크기가 커짐에 따라 기아급수적으로 증가한다. 이러한 문제를 해결하기 위하여 WFlow 알고리즘을 이용한다. 순차적 흐름 구조에서 그래픽 모델에서 사용하였던 MCSP 알고리즘을 사용한다. 추가적으로 MCSP-K 알고리즘도 일반적 흐름 구조 그래픽 모델에서 사용되었다.

이와 더불어 동적 서비스 선택에 관한 여러 관점을 가진 연구들이 있다[4-6]. 동적 서비스 선택에 관한 최근 연구에서는 대부분 비즈니스 프로세스를 생성하는 방법과 시맨틱 정보를 표현하고 사용하는 방법, 개념적 설계를 하는 것을 대부분 다루고 있다. 그러나 개념적 설계와 실용적인 구현 사이에는 차이가 있고 동적 서비스 선택을 하기 위한 특별히 실용적인 지침서를 이용한 설계 방법을 확장할 수 있다.

2.2 Enterprise Service Bus(ESB)

Enterpriser Service Bus(ESB)는 다양한 어플리케이션, 서비스, 자원을 쉽게 통합 연동하고 신뢰성 있는 메시지 통신을 가능하게 하기 위해 데이타 변환, 상황에 알맞은 정확한 라우팅, 메시징을 지원하는 표준 기반의 통합 플랫폼이다[7]. ESB는 메시지 지향 미들웨어(Message Oriented Middleware, MOM), 웹 서비스, 컨텐츠에 알맞은 라우팅을 가능하게 하는 인텔리전트 라우팅(Intelligent Routing), XML 데이타 변환의 주요 기술 요소로 이루어진다. ESB에서 웹 서비스는 추상 앤드포인트(Abstract Endpoint)로 서비스 컨테이너에 등록되며, 서비스 클라이언트는 MOM을 이용하여 서비스 버스를 통하여 다양한 웹 서비스와 상호 작용을 할 수 있게 된다. ESB에서 메시지는 가상 채널을 통해 전송되기 때문에, 메시지는 채널 매니저(Channel Manager)에 의해 감지되며 관리/제어 될 수 있다.

ESB를 구현하기 위해 다양한 표준을 사용할 수 있다. Java Business integration(JBI), J2EE connector Architecture(JCA), Java Management eXtensions (JMX)가 ESB를 구현하기 위한 표준들이다. JBI는 JBI 컨테이너에 있는 다양한 서비스들을 통합할 수 있는 수단을 제공하며, 서비스 간의 상호 연동을 위해 BPEL 엔진과 같은 서비스 엔진(Service Engine)과 다양한 서비스 간의 프로토콜 변환을 가능하게 해주는 바인딩 컴포넌트(Binding Component)가 컨테이너에 플러그 될 수 있다[8]. JBI를 이용하여, 비즈니스 프로세스는 BPEL 엔진에 의해 실행되고 연관된 웹 서비스와 상호작용 할 수 있으며, JBI 컨테이너를 통해 전송되는 여러 메시지들을 관리할 수 있다.

ESB는 Itinerary 기반 라우팅(Itinerary-based Routing)과 컨텐츠 기반 라우팅(Content-based Routing) 기법을 이용한다. 이 기법은 상황에 맞게 알맞은 서비스를 연결시켜 주기 때문에 서비스의 동적 선택과 조합에 중요한 역할을 한다. 그러나, 주어진 비즈니스 도메인이나 서비스 클라이언트의 컨텍스트 정보에 알맞게 서비스를 동적으로 조합하는데 한계점을 지니고 있다.

3. 동적 서비스 선택에 대한 고찰

3.1 동적 서비스 선택

기존 연구는 동적 서비스 선택과 컴포지션에 대한 구분이 불분명하다. 이 장에서는 동적 서비스 선택과 동적 서비스 컴포지션에 대하여 정의를 하고 두 용어의 상호 관계에 대하여 기술한다.

동적 서비스 선택은 비즈니스 프로세스를 실행 시간에 수행을 할 때 각 활동에 해당하는 서비스 컴포넌트를 선택하여 실행한다. 동적 서비스 컴포지션은 실행 시간에 각 활동에 선택된 서비스를 조합하여 비즈니스 프로세스를 실행한다. 동적 서비스 선택이 수행 되면서 비즈니스 프로세스에서 정해진 순서에 맞게 조합이 이루어 진다. 동적 서비스 선택과 컴포지션은 활동을 기준으로 바라보는 관점과 비즈니스 프로세스를 기준으로 바라보는 관점에 따라서 달라진다.

기존에 동적 서비스 선택을 접근하는 방법이 다양한 측면으로 이루어졌다. 기존 접근 방법에 대한 조사를 기반으로 비교 기준을 서비스 선택 결정 시간(*selection decision time*), 목표 서비스 가시성(*target service visibility*), 서비스 적응 지원 정도(*the provision of adaptation support*)로 분류한다.

서비스 선택 결정 시간은 서비스 선택에 대한 결정이 이루어지는 시간을 말한다. BPEL은 설계 시점에서 서비스 선택과 조합이 결정되기 때문에 정적으로 이루어지며, <invoke>을 이용하여 서비스 선택을 명시한다. 그럼에도 불구하고, BPEL은 동적 조합을 위한 기법으로 종종 간주된다. 서비스가 동적으로 선택되어 조합되는 것은 클라이언트의 선호도와 실시간 컨텍스트를 기반으로 서비스를 적응시킬 수 있다는 것 외에 다양한 이점을 가진다. 그러므로, 본 논문에서는 실시간에 서비스 선택이 동적으로 결정되는 것을 목표로 한다.

목표 서비스 가시성은 목표 서비스에 대한 여러 후보들이 미리 결정되어 있는지를 나타낸다. 만약 서비스 후보들이 미리 결정(close)되어 있다면, 서비스 선택은 정의되어 있고 이미 알려진 서비스의 인터페이스를 대상

으로 서비스 선택이 이루어지므로 서비스 후보들이 미리 결정되지 않는 경우에 비하여 쉽다. 서비스 후보들이 미리 결정되어 있지 않다면(open), 목표 서비스에 대한 후보들을 서비스 저장소에서 실시간으로 검색해야 하고, 이들의 인터페이스를 신택틱(syntactic) 정보뿐 아니라 시맨틱 정보를 이용하여 적절한 서비스를 찾아야 하기 때문에 서비스를 선택하여 조합 하는데 어려움이 발생할 수 있다. 본 논문에서는 미리 정해져 있지 않고 실시간에 배포된 목표 서비스를 대상으로 조합하는 것을 목표로 한다.

서비스 적응 지원 정도는 목표 서비스들이 주어진 서비스 요청에 맞게 적응될 수 있는지에 대한 방법이 있는지를 나타낸다. 이 기준은 적응이 지원되지 않은 경우, 수동적인 적응이 지원되는 경우, 자동적인 적응이 지원되는 경우로 분류된다. 적응성(adaptability)는 목표 서비스 가시성에 크게 의존한다. 즉, 만약 가시성이 미리 정의되어 있지 않다면(open), 새롭게 배포된 웹 서비스를 발견하고 서비스 요청을 만족시켜야 하기 때문에, 서비스 적응은 필수적이다. 완전히 자동적으로 서비스 적응이 이루어지는 것은 현재 SOA 기술과 도구로 불가능하다. 그러므로, 본 논문에서는 수동적으로 서비스 적응을 지원하는 것을 목표로 한다. 즉, 서비스 적응 로직은 서비스 컨텍스트의 시맨틱 정보를 고려하여 사람에 의해 결정될 수 있다.

위와 같은 기준들을 기반으로, 동적 서비스 선택에 대한 본 논문의 접근 방법은 그림 1과 같다. 즉, 본 논문에서는 미리 정해져 있지 않은 목표 서비스(open)에 대해, 수동적(manual)으로 서비스를 동적 조합 하는 방법을 다룬다.

3.2 본 논문의 접근 방법

본 논문의 접근 방법을 이용하여, 서비스 클라이언트는 다양한 후보 서비스를 동적으로 발견함으로써 목표

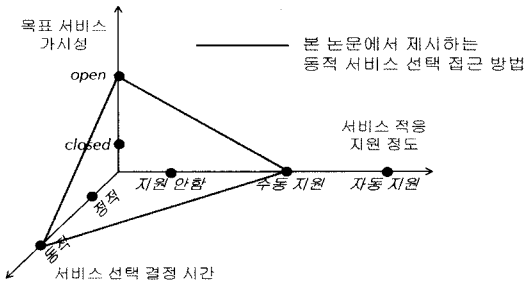

그림 1 본 논문의 동적 서비스 선택 범위

서비스를 조합한다. 그림 2는 서비스가 동적으로 선택되는 과정을 추상적으로 보여준다.

그림 2의 비즈니스 프로세스는 3개의 활동(activity)으로 구성되며, 각 활동은 적절한 서비스 컴포넌트를 호출함으로써 실행된다. 활동1과 활동3은 호출하는 서비스 컴포넌트가 정해져 있지만 활동2의 경우는 실행될 수 있는 서비스 컴포넌트가 하나 이상 존재한다. 그러므로, 다양한 서비스 클라이언트와 각 클라이언트의 상황에 맞게 가장 적절한 서비스 컴포넌트를 호출해야 한다.

동적 서비스 선택을 해결하기 위해, 본 논문에서 동적 선택 핸들러(Dynamic Selection Handler, DSH) 설계 기법을 제안한다. DSH는 가장 적절한 서비스 컴포넌트를 결정하여 호출하는 역할을 가진다. DSH는 특정 비즈니스 프로세스와 목표 서비스와는 독립적이게 설계되며, 이를 이용하여 동적 서비스 선택은 서비스 클라이언트와 서비스 제공자 모두에게 투명하게 이루어진다.

본 논문에서는 DSH를 구현하기 위한 플랫폼으로 ESB를 이용한다. ESB는 비즈니스 프로세스와 목표 웹 서비스 간의 메시지를 직접적으로 연결시켜 줌으로써 메시지를 전송해 준다. 그러므로, 서비스 선택이 동적으로 이루어질 수 있다. 만약 새로운 웹 서비스가 발행된

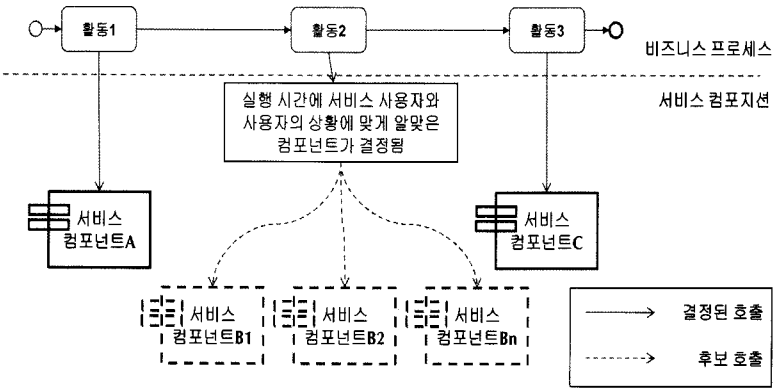

그림 2 비즈니스 프로세스 내의 서비스 동적 선택

다면, DSH는 동적으로 이 서비스를 발견하여 새로운 서비스들을 선택 할 수 있게 된다.

4. 동적 선택 핸들러 설계 기법

이 장에서는 실시간에 알맞은 서비스 선택을 결정하고, 각 선택된 서비스를 호출하기 위한 DSH를 설계한다. DSH를 설계 하기 위해 서비스 인터페이스를 BPEL에서 명세 된 WSDL 인터페이스와 서비스컴포넌트에서 실체화된 WSDL 인터페이스로 분류한다. 그림 3은 두 종류의 WSDL 인터페이스들간의 관계를 보여준다.

BPEL에서 명세 된 WSDL 인터페이스는 클라이언트 관점에서 명세된 서비스 인터페이스로서 실체화된 서비스 컴포넌트에 대한 앤드 포인트 정보를 포함하지 않는 추상 인터페이스이다. BPEL은 <invoke>를 이용하여 클라이언트가 명세한 WSDL 인터페이스를 호출한다. BPEL에서 WSDL 인터페이스에는 서비스 컴포넌트를 호출할 때 가장 적합한 서비스 컴포넌트를 동적으로 바인딩 하기 때문에 호출되는 서비스 컴포넌트에 관한 구현 정보는 포함하지 않는다. 서비스 컴포넌트의 WSDL 인터페이스는 실체화되어 있기 때문에 앤드 포인트 정보를 제공한다. 서비스 컴포넌트의 WSDL 인터페이스는 이미 배포된 서비스 컴포넌트를 위해 정의 되거나 레거시 어플리케이션을 랩핑하기 위한 목적으로 정의된다.

두 WSDL 인터페이스를 분리함으로써 비즈니스 프로세스는 실제 호출될 서비스 컴포넌트에 맞게 재구성되거나 재 배포될 필요가 없다. 즉, 서비스 선택은 비즈니스 프로세스 배포 또는 서비스 컴포넌트 배포와는 독립적으로 결정된다.

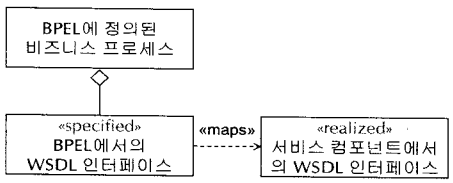

그림 3 서비스 인터페이스들간의 관계

서비스 구현 계층은 ESB에 앤드 포인트를 통하여 서비스 컴포넌트들과 연결 되어 있다. 서비스 버스 계층은 여러 서비스와 클라이언트 프로그램, BPEL 엔진, 서비스 저장소, DSH 등을 통합하여 신뢰성 있는 메시지 통신을 가능하게 한다. 현재 SOA 표준으로는 ESB가 여러 프로그램간의 메시지 통신을 가능하게 하는 역할을 수행한다.

ESB에 새롭게 추가된 DSH는 기존의 컴포넌트나 서비스간의 실행 및 통신을 감지하여 적합한 서비스 컴포넌트를 호출하는 역할을 한다.

DSH는 Invocation Listener, Service Selector, Service Binder 및 Interface Transformer의 네 개의 컴포넌트로 구성되어 있고 서비스를 선택하는데 사용하는 결정 규칙 데이타 베이스와 인터페이스 불일치를 해결하기 위한 인터페이스 규칙 저장소를 사용한다.

클라이언트는 비즈니스 프로세스에서 명세 된 서비스를 호출한다. BPEL은 여러 활동으로 구성되어 있고 여러 서비스 컴포넌트를 수행하여 클라이언트가 원하는 서비스를 제공한다. DSH는 BPEL를 수행하기 위하여 클라이언트가 보낸 메시지를 모니터 하여 클라이언트가 선호하는 후보 서비스를 선택하여 실체화된 서비스 컴포넌트에 맞게 연결한다. DSH의 요청에 따라 해당 서비스를 수행한다.

클라이언트는 JMS API를 이용하여 비즈니스 프로세스에서 명세 된 서비스를 호출하기 위하여 메시지를 생성하여 ESB에 보낸다. DSH에 있는 Invocation Listener는 클라이언트가 비즈니스 프로세스를 수행하기 위해 생성한 메시지를 감지하여 Service Selector로 전달한다. ESB 상에서 감지된 메시지를 Service Selector는 결정 규칙을 이용하여 후보 서비스를 동적으로 서비스

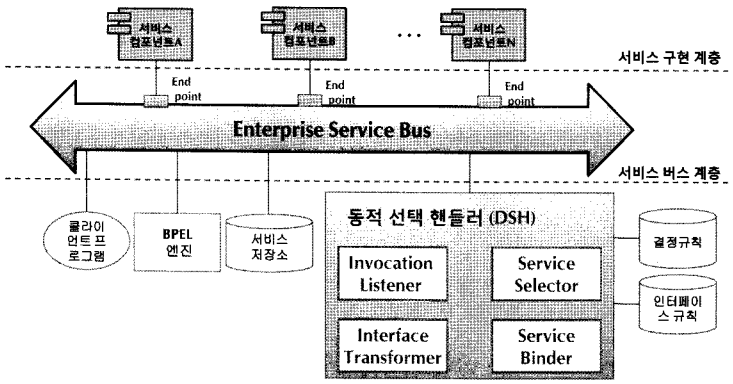

그림 4 ESB상에서의 동적 선택 핸들러 구조

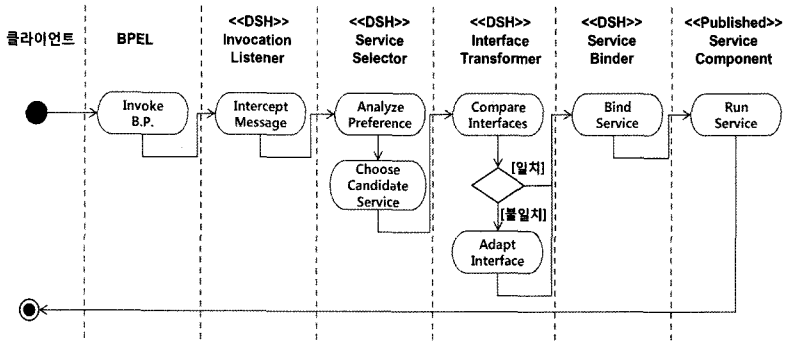

그림 5 DSH 기능 흐름도

저장소에서 선택한다. 결정 규칙은 클라이언트의 선호에 따른 정보가 미리 정의되어 있거나 입력을 받는다. 선택된 후보 서비스는 비즈니스 프로세스 수준에서 명세 된 서비스와 비교를 하기 위하여 Interface Transformer로 보내지게 되고 비즈니스 프로세스 수준에서 명세 된 서비스와 후보 서비스 인터페이스를 비교하여 일치하였을 경우는 후보 서비스를 바인딩하고 불일치 한 경우에는 인터페이스 규칙에 기반하여 어댑터를 이용하여 서비스를 바인딩 한다. Service Binder는 실제적인 앤드 포인트가 있는 서비스 컴포넌트를 호출하는 역할을 한다. 호출된 서비스 컴포넌트 결과는 ESB를 통하여 클라이언트로 전달된다.

DSH의 Service Selector 컴포넌트와 Interface Transformer 컴포넌트에 적절한 디자인 패턴을 사용하여 설계한다. DSH의 각 컴포넌트가 DSH 내의 다른 컴포넌트 또는 ESB와 어떻게 연관관계를 가지는지 기술할 것이다.

4.1 Invocation Listener

클라이언트는 비즈니스 프로세스를 수행하기 위하여 메시지를 ESB에 보낸다. Invocation Listener는 비즈니스 프로세스에서 동적 서비스 선택이 필요한 서비스 컴포넌트를 호출하는 메시지를 감지하는 역할을 한다. 서비스 선택은 실행 시간에 인식이 되고 관리가 되기 때문에 Invocation Listener는 실행 시간에 발생하는 메시지를 모니터한다. 이 컴포넌트는 관련된 Service Selector 컴포넌트에 인식된 서비스 컴포넌트 메시지를 넘겨준다.

BPEL 엔진이 수행할 수 있도록 클라이언트가 생성한 메시지를 ESB에서 해당 메시지를 큐에 전달 되기 전에 Invocation Listener가 메시지를 가로채는 역할을 한다. 클라이언트가 생성한 메시지에는 서비스 이름과 앤드 포인트에 관한 내용을 포함하고 있고 *MessageListener*를 상속받은 Invocation Listener는 해당 서비스 이름과 앤드 포인트의 내용이 포함되어 있는 메시지를 ESB 상에서 받아 온다.

4.2 Service Selector

Service Selector는 후보 서비스로부터 적합한 서비스를 결정하는 역할을 한다. 서비스 컴포넌트 결정시 클라이언트 선호도와 컨텍스트를 분석하기 위한 결정 규칙이 사용된다. 이 컴포넌트는 데이타 베이스에 있는 컨텍스트 정보를 관리하고, 미리 정의된 규칙을 기반으로 컨텍스트 정보를 해석한다. 시맨틱 정보를 분석하기 위하여 [9]와 [10] 같은 방법을 적용할 수 있다. 이 논문에서는 시맨틱 분석으로 얻은 컨텍스트 분석 결과와 입/출력 매개변수를 이용하여 서비스를 결정한다.

Service Selector는 Invocation Listener가 쉽게 사용하도록 상위 수준 인터페이스를 제공한다. 그림 6은 Service Selector 컴포넌트가 Service Binder 컴포넌트와 Interface Transformer 컴포넌트와 어떻게 상호 작용하는지를 보여준다.

Service Selector 컴포넌트는 세 컴포넌트들과 메시지 상호 작용하면서 기능을 수행한다. 첫 번째 상호작용인 (a)를 통해서, Invocation Listener 컴포넌트는 비즈니스 프로세스의 각 활동을 수행하기 위하여 클라이언트가 생성한 메시지로부터 명세된 서비스 인터페이스, 프로파

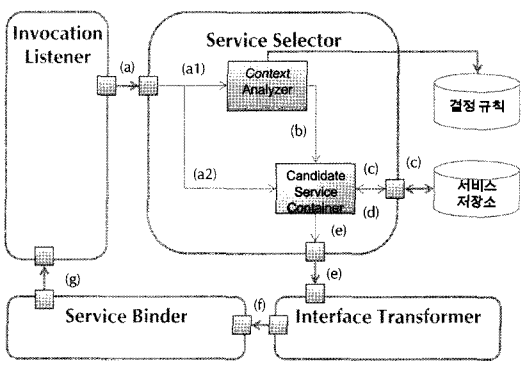

그림 6 Service Selector 아키텍처

일 정보, 서비스 검색 조건 정보를 Service Selector 컴포넌트에게 보낸다.

프로파일 정보와 서비스 검색 조건을 *ContextAnalyzer* 클래스와 *CandidateServiceContainer* 클래스로 위임(a1, a2)하여 클라이언트 선호도를 분석한 후 결과를 *CandidateServiceContainer* 클래스에 전달한다(b). *CandidateServiceContainer* 클래스는 프로파일 분석 결과를 이용하여 서비스 저장소에서 후보 서비스를 검색하고 발견하는 메시지를 UDDI(Universal Description, Discovery and Integration)서비스 저장소로 보낸다(c). UDDI에서 검색된 후보 서비스 결과를 *CandidateServiceContainer* 클래스로 보내며(d), *CandidateServiceContainer* 클래스는 서비스 저장소에서 찾은 정보를 명세 수준의 서비스 인터페이스와 후보 서비스 인터페이스를 분석을 위한 Interface Transformer 컴포넌트에 메시지를 전달한다(e). Interface Transformer 컴포넌트에서 인터페이스 불일치를 해결한 후 메시지를 Service Binder 컴포넌트에게 전달한다(f). Service Binder 컴포넌트는 적합한 응답 메시지를 생성하여 Invocation Listener 컴포넌트에게 전달한다(g).

후보 서비스를 검색하고 결과를 다른 컴포넌트로 전달하여 검색된 결과를 해당 컴포넌트에서 처리하지 않고 다른 컴포넌트에게 전달하여 수행할 수 있도록 하기 위하여 Delegation 패턴을 이용하여 추출된 결과를 다른 컴포넌트에게 전달한다. 이렇게 함으로써, 한 컴포넌트가 수행할 기능이 집중되는 것을 방지하는 이점이 있다.

그림 7은 Service Selector가 Delegation 패턴을 이용하여 Interface Transformer을 호출하여 기능을 수행하는 절차를 보여주는 시퀀스 다이어그램이다. *ContextAnalyzer* 클래스에서 분석된 *preferenceType* 조건에 맞는 서비스 명을 UDDI 서비스 저장소에 전달하고 조건에 맞는 후보 서비스를 획득하여 *CandidateFindingServiceContainer* 클래스에 해당 서비스를 전달한다. 후보 서비스를 찾는 메소드인 *getFindCandiateService* 가 호출되지만 실제적인 수행은 *getCandiateService* 메소드에서 일어나게 된다. *CandidateFindingServiceContainer* 클래스는 인터페이스 불일치를 알아보기 위하여 Interface Transformer 클래스에 명세된 인터페이스와 후보 서비스를 전달하여 인터페이스 간의 정보가 서로 일치하는지에 대한 결과를 얻게 된다.

4.3 Service Binder

Invocation Listener 컴포넌트는 서비스 이름과 앤드포인트정보를 포함한 클라이언트가 생성한 메시지 요청을 인지하였지만 BPEL 엔진이 메시지를 수행하기 위한 요청을 해당 큐에 보내지 않는다. Service Binder는 Service Selector에 의해 결정된 하나의 서비스를 기반으로 Interface Transformer 컴포넌트에서 수정된 정보를 전달 받는다. DSH의 Service Binder는 요청 메시지를 생성하여 Invocation Listener에 전달하는 역할을 한다.

호출될 서비스 컴포넌트의 WSDL 인터페이스에서 수행되어야 하는 메소드를 선택한다. WSDL 인터페이스에서 메소드는 WSDL 1.1을 기준으로 하여 <portType>의 서브 엘리먼트인 <operation>에 명시가 되어 있다. 해당 메소드의 입력이 다른 경우는 <types> 엘리먼트에 정의되어 있는 name 속성에 해당하는 형태로 타입을 변환하여 입력을 Interface Transformer 컴포넌트가 메시지를 생성한다. 생성된 메시지를 XML 형태로 변환하여 Invocation Listener 컴포넌트에 전달하게 된다. 전달된 메시지는 해당 큐에 다시 보내지게 되고 전달된 메시지는 호출될 서비스 컴포넌트에 보내지게 된다.

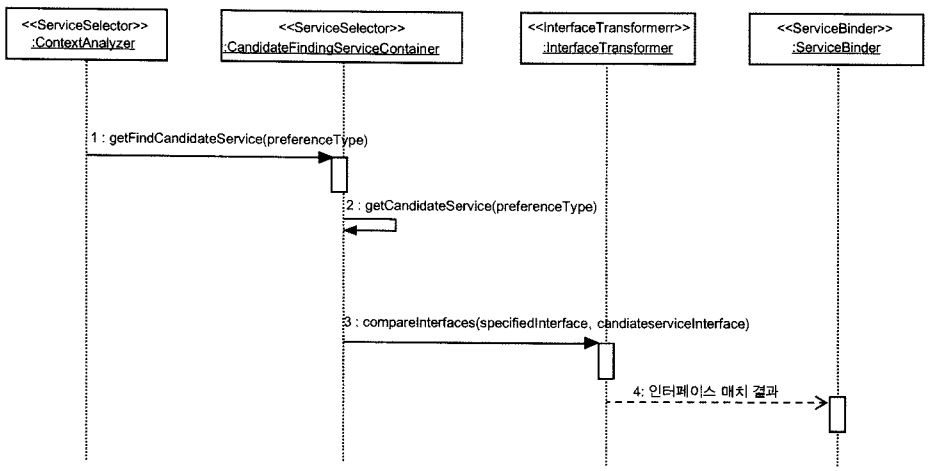

그림 7 Service Selector 시퀀스 다이어그램

4.4 Interface Transformer

서비스가 제공해야 하는 기능과 클라이언트 선호도 정보를 기반으로 적절한 서비스가 결정되었지만 요청된 서비스 인터페이스 정보는 Service Selector에 의해 결정된 웹 서비스 인터페이스 정보와 불일치 할 수 있다. DSH의 Interface Transformer 컴포넌트는 명세 수준의 WSDL과 실체화 수준의 WSDL간의 불일치를 해결하여 적응시키는 역할을 한다.

Service Selector에서 전달받은 후보 서비스 정보(a)를 BPEL의 WSDL 인터페이스 정보와 서비스 컴포넌트의 WSDL 인터페이스 정보를 *InterfaceRuleAdapter* 클래스에서 인터페이스 규칙 저장소에서 미리 정의된 규칙을 기반으로 불일치한 경우에 적합한 Adapter를 결정하여 *InterfaceTransformer* 클래스에게 전달한다(b1). *InterfaceRuleAdapter* 클래스에서 불일치가 발생하지 않는 경우는 Service Binder 컴포넌트에 웹 서비스 인터페이스 정보를 전달한다(b2). *InterfaceTransformer* 클래스에서 인터페이스 불일치를 해결한 정보를 Service Binder 컴포넌트에 전달한다(c).

인터페이스 적응을 하기 위하여 인터페이스 규칙 저장소에 여러 형태의 WSDL 인터페이스 불일치를 해결하는 규칙이 정의가 되어 있어야 한다. (1) 메소드 이름이 다른 경우, (2) 속성 이름이 다른 경우, (3) 데이타 타입이 다른 경우, (4) 추가 매개변수가 필요한 경우, (5) 단일 매개변수 값이 여러 매개변수가 분할되어 할당되는 경우, (6) 여러 매개변수 값이 한 매개변수 값으로 할당되는 경우, (7) 선택적 매개변수이기 때문에 목표 인터페이스에서 매개 변수가 더 이상 필요하지 않는 경우를 고려한 적당한 규칙을 정의 한다.

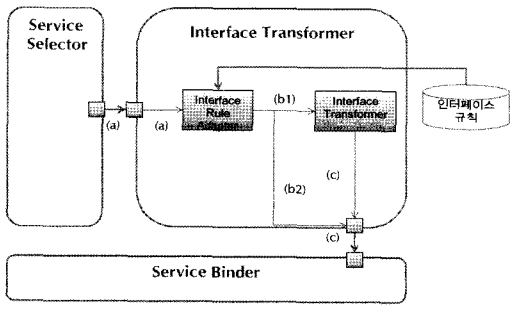

그림 8 Interface Transformer 아키텍처

클래스의 인터페이스를 클라이언트가 원하는 형태의 인터페이스로 변환하는 경우 BPEL 에서의 인터페이스와 서비스 컴포넌트의 인터페이스가 서로 일치하지 않는 클래스들을 동작시킨다. 인터페이스를 적응시키기 위해 여러 종류의 어댑터를 구현하기 위하여 Interface Transformer 컴포넌트에 Adapter 패턴을 이용하여 설계한다.

그림 9는 Interface Transformer 개발 단계에서 가능한 여러 형태의 어댑터를 고려하고 target 클래스와 adaptee 클래스 사이의 일치하지 않는 인터페이스를 해결하기 위하여 Adapter 패턴을 적용한 클래스 다이어그램이다.

인터페이스 불일치가 발생하면 Interface Transformer는 호출되는 서비스 인터페이스를 가지고 미리 선언된 규칙을 이용하여 실제화 수준의 인터페이스로 변환을 한다. 그림 10은 인터페이스 불일치를 해결하는 알고리즘을 보여준다.

그림 10은 *comapreInterface()*와 *adaptInterfaceMismatch()* 메소드를 가진다. 첫 번째 메소드는 인터페이스 불일치 유무를 확인한다. 결과값이 참일 경우는 인터페이스 불일치가 발생한 것이다. 두 번째 메소드는 인터페이스 불일치를 해결하는 메소드이다. *adaptInterfaceMismatch()*에서 미리 정의된 규칙에 기반을 하여 인터페이스 불일치 형태가 결정되고 해결한다. *If-else* 문은 불일치 형태에 따른 해당 인터페이스와의 연결 방법을 보여준다.

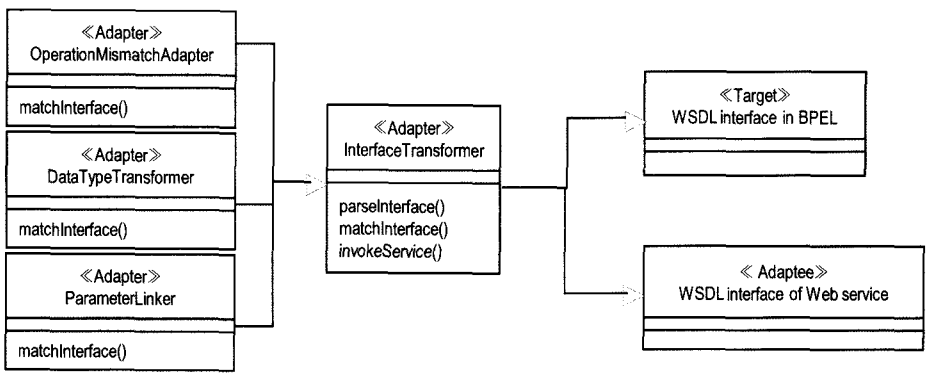

그림 9 Interface Transformer 클래스 다이어그램

```
public boolean compareInterface ( OperationSignature original, OperationSignature target) {
    boolean result = false;
    if ( original.size() != target.size()) {
        result = true;  // 매개변수 수가 다른 경우 불일치 발생
    } else {
        for ( int i = 0 ; i < original.size() ; i++ ) {
            if ( !original.getElement(i).toString().equals (target.getElement(i).toString()) ) {
                result = true; // 속성이 다른 경우 불일치 발생
            }
        }
    } return result;
}
public String adaptInterfaceMismatch (OperationSignature original, OperationSignature target) {
    OperationSignature matched = new OperationSignature();
    MismatchCase mismatch = AdaptationRule.getAdaptationRule();
    if ( MismatchCase.type == 1 ) {              // 메소드 이름이 다른 경우
        matched.setMethodName (original.getMethodName() );
    } else if ( MismatchCase.type == 2)          // 매개변수 이름이 다른 경우
        matched.setElement(j, original.getElement(i));
    } ...
    return matched;
}
```

그림 10 Interface Transformer 알고리즘

5. ESB를 이용한 동적 선택 핸들러 구현

5.1 호텔 예약 시나리오

DSH 구현의 실행 가능성을 보여주기 위하여, 호텔 예약 시나리오를 이용하여 사례 연구를 수행한다. 이 서비스는 클라이언트의 선호도를 기반으로 후보 호텔들을 검색하고, 이 후보들로부터 가장 적절한 호텔을 결정한다. 그림 11은 종합 호텔 예약 서비스 시스템(Total Hotel Reservation Service System, THRSS)의 전체 시나리오를 Business Process Modeling Notation (BPMN) 표기법을 이용하여 보여준다.

여행자로부터 받은 ID와 비밀번호 정보를 이용하여 데이타베이스에 저장되어 있는 여행자의 히스토리 정보를 검색한다. 이 정보는 여행자의 선호도(주요 여행 목적, 선호하는 호텔 등급, 객실 타입 등)을 식별하는데 사용된다. 그리고, 분석된 클라이언트 선호도 결과를 기반으로 THRSS는 가장 적절한 호텔 검색 서비스를 찾는다. 여러 호텔 검색 서비스가 검색되면 클라이언트는 적절한 호텔 검색 서비스를 선택하고, 선택된 호텔 검색 서비스는 조건에 맞는 호텔과 각 호텔의 WSDL 주소를 반환한다. 반환된 호텔에 이용 가능한 객실이 있으면 상세 정보를 여행자에게 보여주고, 여행자는 상세 정보를 기반으로 호텔을 선택하고 예약할 수 있게 된다. 이런 활동들은 모두 클라이언트가 호텔 검색 서비스를 요청하였을 때 동적으로 실행된다.

5.2 동적 서비스 선택을 위한 아키텍처

시나리오를 기반으로, 그림 12와 같이 '클라이언트 프로파일 분석하기(*Analyze Client Profile*)', '호텔 검색하기(*Find Hotels*)', '호텔 정보 얻어오기(*Get Hotel Information*)', '호텔 예약하기(*Reserve Hotel*)' 4개의 활동으로 이루어진 비즈니스 프로세스를 설계한다.

비즈니스 프로세스의 각 활동은 적합한 서비스 컴포넌트를 호출함으로써 수행된다. 활동 1(클라이언트 프로파일 분석하기)는 THRSS 시스템 내부에 EJB로 구현된 "클라이언트 프로파일 분석기"를 호출함으로써 클라이언트가 입력한 ID와 데이타베이스에 저장된 히스토리 정보를 이용하여 클라이언트 프로파일 및 선호도를 분석한다. 그 외의 활동들은 외부 서비스를 호출함으로써 각 기능들을 수행한다. 활동 2(호텔 검색하기)는 THRSS 시스템 외부에서 구현된 2개의 후보 서비스 컴포넌트(호텔 검색 서비스 A와 호텔 검색 서비스 B)가 있다. 분석된 클라이언트 프로파일 정보를 기반으로 클라이언트에게 가장 적합한 호텔을 검색해 줄 수 있는 호텔 검색 서비스를 결정하고, 선택된 호텔 검색 서비스를 이용하여 클라이언트 조건에 알맞으면서 사용 가능한 호텔들을 검색해준다. 활동 3(호텔 정보 얻어오기)는 호텔 정보 서비스가 검색한 호텔 목록과 WSDL 정보를 이용해서 각 호텔들을 호출하여 호텔 정보를 얻어온다. 마지막으로 활동 4(호텔 예약하기)에서는 클라이언트가 선택한 호텔 서비스를 호출함으로써 클라이언트가 선택한 객실을 예약한다.

본 사례 연구에서의 동적 서비스 선택은 활동 2(호텔 검색하기)를 위한 두 개의 서비스 컴포넌트 중 한 컴포넌트를 결정하고 호출하는데 중점을 둔다. 활동 2(호텔 검색하기)에 해당하는 적절한 서비스 컴포넌트를 찾기 위한 DSH는 그림 13과 같이 ESB 기반으로 설계된다.

비즈니스 프로세스에서 활동 2를 수행하기 위해 (1) 서비스 컴포넌트를 호출하는 메시지를 동적 선택 핸들

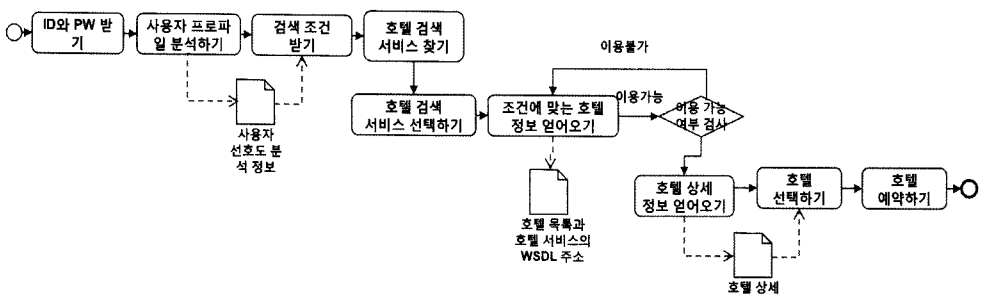

그림 11 BPMN 표기법을 이용한 호텔 예약 시나리오

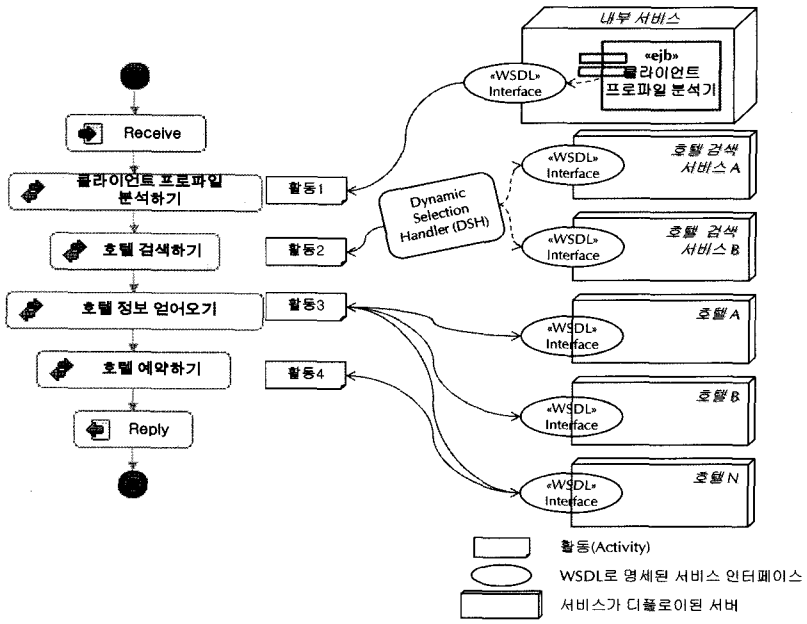

그림 12 비즈니스 프로세스와 서비스 컴포넌트 아키텍처

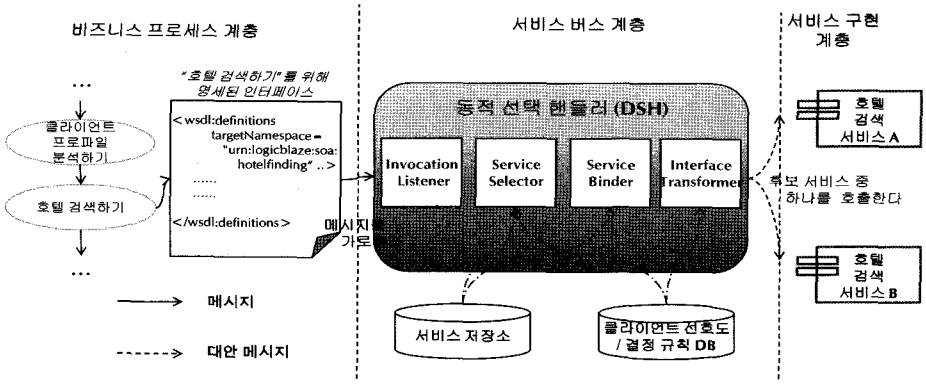

그림 13 ESB 기반의 DSH 요소들

러가 가로챈다. 동적 선택 핸들러는 가로챈 메시지를 이용하여, (2) 클라이언트의 프로파일 및 선호도를 분석하고 적절한 서비스를 선택하고, 선택된 서비스에 맞게 메시지를 수정한다. (3) 수정된 메시지는 메시지 큐로 보내지고, (4) 선택된 서비스(호텔 검색 서비스)를 실제 호출하여 호텔을 검색하는 서비스 기능을 수행한다.

5.3 구현

Celtix[11], ServiceMix[12], Open ESB[13]은 ESB를 지원하는 툴이다. 본 사례 연구에서는 DSH를 구현하기 위한 플랫폼으로 오픈 소스 기반의 JBI 표준을 준수한 ServiceMix를 선택한다. ServiceMix에 DSH의 4개 컴포넌트, 비즈니스 프로세스를 실행하기 위한 BPEL과 WSDL 인터페이스를 작성한다. 서비스 컴포넌트의 WSDL 인터페이스는 여러 개가 존재하여 동적 서비스 선택 기능성을 구현하는데 사용된다.

서비스 클라이언트는 그림 14와 같이 명세 된 호텔 예약 비즈니스 프로세스를 클라이언트를 위한 WSDL 인터페이스를 통하여 호출한다. 그림 14의 비즈니스 프로세스는 BPEL 표준을 준수하기 때문에, BPEL 엔진은 동적 서비스 선택을 위한 비즈니스 프로세스 호출임을 인식하지 않고 프로세스를 수행하게 된다. 그리하여, 서비스 클라이언트를 위한 비즈니스 프로세스를 수정하지 않고 동적 서비스 선택을 가능하게 하는 DSH을 호출하여 수행할 수 있다.

```
<bpel:process name="HotelReservationProcess"
  targetNamespace="urn:logicblaze:soa:hotelreservation"
  xmlns:tns="urn:logicblaze:soa:hotelreservation"
  xmlns:ca="urn:logicblaze:soa:hotelfinding"
  ....
  <bpel:variables>
     <bpel:variable name="request"  messageType="tns:reserveHotelRequest" />
     <bpel:variable name="response"  messageType="tns:reserveHotelResponse" />
     <bpel:variable name="ca-findHotel-request"  messageType="ca:findHotelRequest" />
     <bpel:variable name="ca-findHotel-response"  messageType="ca:findHotelResponse" />
     ....
  <bpel:variables>

  <bpel:sequence>
      ...
     <bpel:sequence>
        ...
       <bpel:invoke name="service" partnerLink="HotelFinding"
          portType="ca:HotelFinding" operation="findHotel"
          inputVariable="ca-findHotel-request"
          outputVariable="ca-findHotel-response"/>
          ...
     </bpel:sequence>
    ....
</bpel:process>
```

그림 14 BPEL로 명세된 호텔 예약 프로세스

```
public void run() {
    try {
      //요구 생성
      String request =
        " <reserveHotelRequest xmlns='urn:logicblaze:soa:hotelreservation'>\n" +
        " <userID>gemini79</userID>\n" +
        " <sDate>2007-03-01</sDate>\n" +
        " <duration>5</duration>\n" +
        " <area>San Francisco</area>\n" +
        " <roomType>single</roomType>\n" +
        " </reserveHotelRequest> ";

      //TextMessage  out 메시지 생성
        TextMessage out = requestor.getSession().createTextMessage(request);

      //TextMessage in 메시지 요청
        TextMessage in = (TextMessage) requestor.request(outQueue, out);

      // in 메시지 예외처리
        if (in == null) {
        System.out.println("Response timed out.");
        }

      // in 메시지 출력
       else {
         System.out.println("Response: " + in.getText());
       }
    } catch (Exception e) {
      e.printStackTrace();
    }
}
```

그림 15 JMS 클라이언트 - JMSClient.java

BPEL에서 *HotelReservationProcess*가 *findHotels* 인터페이스를 호출할 때, *InvocationLisenter* 클래스(Hotel-Binding.java)는 서비스 호출을 인식하고 동적 컴포지션을 수행하기 위해 메시지를 가로챈다. 그림 16은 *InvocationLisenter* 구현의 일부를 보여준다. 이 리스너 클래스(*InvocationLisenter*)는 리스너에서 인식된 메시지를 처리하는 기능을 수행하는 JBI 스팩의 *MessageExchangeListener* 인터페이스를 구현한다.

클라이언트는 요청하는 메시지를 그림 15와 같이 *String* 형태의 *request*를 생성을 한다. 생성된 메시지를 *TextMessage*의 인스턴스인 *out*을 생성한다. *out*은 생성된 요청을 해당 큐에 보내는 역할을 한다. *Requestor* 클래스의 *request()*을 호출하여 *TextMessage*의 인스턴스인 *in*에 결과값을 할당한다. *in*은 해당 큐에서 처리된 메시지를 받는 역할을 한다. 시간 지연 및 선택

```
public class HotelFinding extends ComponentSupport
                          implements MessageExchangeListener {
    public HotelFinding() {
      // JMS, Listener, Creation
       setService(new QName("urn:logicblaze:soa:hotelfinding",
                          "HotelFindingService"));
       setEndpoint("HotelFinding");
    }

    public void onMessageExchange(MessageExchange exchange)
                                              throws MessagingException {
      // Listening Invocation Message
      if (inOut.getStatus() == ExchangeStatus.DONE) {
        return;
      } else if (inOut.getStatus() == ExchangeStatus.ERROR) {
        return;
      }

      try {
        Document doc =
  (Document) new SourceTransformer().toDOMNode(inOut.getInMessage());
        String userID  = textValueOfXPath(doc, "//*[local-name()='userID']");
        String sDate   = textValueOfXPath(doc, "//*[local-name()='sDate']");
      ...
      // invoke the service router
      findHotelServiceResult = sr.findHotelFindingService(profileType, os, rs);
      ...
```

그림 16 Invocation Listener - HotelFinding.java

불가능한 경우는 *in*의 값이 *null*인 상태가 된다.

이 리스너는 3개의 인수인 클라이언트 프로파일 타입(profileType), 리스너가 인식한 오퍼레이션 정보(os), 예약 관련 정보(rs)를 이용하여 Service Selector를 호출한다. 클라이언트 프로파일 타입 정보는 클라이언트가 원하는 호텔 정보에 대한 상세화 정도를 결정하여 선호도 정보를 포함한다. MySql을 이용하여 *Preference* 테이블에서 클라이언트 정보와 프로파일 타입 및 지역정보를 쿼리를 실행하여 선호도와 관련된 정보를 수집한다. 리스너가 인식한 오퍼레이션 정보는 WSDL 인터페이스에 명세 수준으로 정의된 오퍼레이션 정보를 나타낸다. 리스너가 Service Selector를 이용하여 클라이언트의 프로파일 정보에 맞게 호텔 검색 서비스를 선택하는 *findHotelfindingService*를 호출할 때, 그림 17과 같은 절차로 동적 서비스 선택이 수행된다.

*ServiceSelector*는 *EventListener*에서 넘겨받은 오퍼레이션 정보를 이용하여, 클라이언트 프로파일 타입및

ReservationType 클래스를 분석하여 *UserPreference-Analyzer* 클래스의 *analyzeClientPreference()*를 호출한다. *ServiceSelector*는 클라이언트 프로파일 타입에 맞는 호텔 검색 서비스를 찾기 위해 *CandidateFinding-ServiceContainer* 클래스의 *HotelFindService()*를 호출한다. 결정된 선호도가 1인 경우에는 *Hotel Finding Service A*가 선택이 되고 2인 경우에는 *Hotel Finding Service B*가 선택이 된다. 이 때, *CandidateFinding-ServiceContainer*는 서비스 저장소에 등록되어 있는 서비스 정보를 대상으로 검색한다. 적절한 호텔 검색 서비스가 선택되면, *ServiceSelector*는 선택된 호텔 검색 서비스의 오퍼레이션과 원래 리스너가 인식한 오퍼레이션 정보가 일치하는지 확인한다. 두 오퍼레이션 정보가 불일치하면 *InterfaceTransformer* 클래스의 *adaptInterfaceMismatch()*를 호출하고, 그렇지 않으면 선택된 호텔 검색 서비스를 호출한다.

이 시나리오에서는 호텔 검색 서비스 중에 클라이언트가 선택한 프로파일 타입에 따라 *Hotel Finding Service A*가 선택되었으며, Service Selector 컴포넌트는 선택된 서비스 컴포넌트에 적응된 인터페이스와 서비스의 앤드 포인트와 바인딩 정보를 포함하는 SOAP 메시지를 만들고, 이 메시지를 해당 앤드 포인트로 전송한다.

6. 평가

이번 장에서는 제안된 동적 서비스 선택 기법을 기존 연구를 비교함으로써 제안된 기법을 평가한다.

본 연구의 동적 서비스 선택 기법과 기존 연구를 비교하기 위한 평가 항목을 도출하기 위하여, 3.2절에서 식별한 동적 서비스 선택을 분류하는데 적용된 3가지 기준을 이용한다. "서비스 선택 결정 시간"관점에서 서비스를 동적으로 선택하기 위해서는 클라이언트의 시맨틱 정보와 서비스 자원을 고려하여 실시간에 서비스 선택이 가능해야 하며, 동적으로 서비스를 선택함으로써 생길 수 있는 오버헤드도 고려해야 한다. 그러므로, "실시간 서비스 선택", "시맨틱 정보 제공", "다양한 서비스 자원 고려", "프로세스 재배포로 인한 오버헤드 고려" 평가 항목이 도출되었다. "목표 서비스 가시성" 관점에서 동적 서비스를 지원하기 위해서는 실시간에 비가용 서비스를 발견하여 새로운 서비스로 대체해야 하므로, "비가용 서비스 발견"이 도출되었다. 마지막으로, "서비스 적응 지원 정도" 관점에서는 서비스가 동적으로 선택될 때 서비스 클라이언트의 요청을 만족시킬 수 있도록 서비스 적응이 필요하며 이로 인해 서비스 클라이언트에게 적합한 서비스를 제공할 수 있게 된다. 그러므로, "개인화 지원"과 "서비스 적응 지원"이 평가 항목으로 도출되었다.

표 1은 도출한 평가 항목별로 기존 연구와 본 연구에서 제안한 기법의 지원여부를 보여준다.

Fujii의 연구에서는 시맨틱 기반의 동적 서비스 컴포지션 시스템을 제안하였다. 제안된 시스템은 시맨틱 정보를 분석하여 서비스를 조합하여 수행하였다. 이미 정의된 서비스를 기반으로 하여 워크플로우를 생성하여 수행하였기 때문에 가용하지 않는 서비스 발견에 대한 고려가 부족하였다.

Penta의 연구는 선호도에 따른 서비스 조합을 동적으로 바인딩하기 위한 프레임워크를 제안하였다. 제안된 프레임워크는 수행 전 바인딩과 실행 시간 바인딩을 고

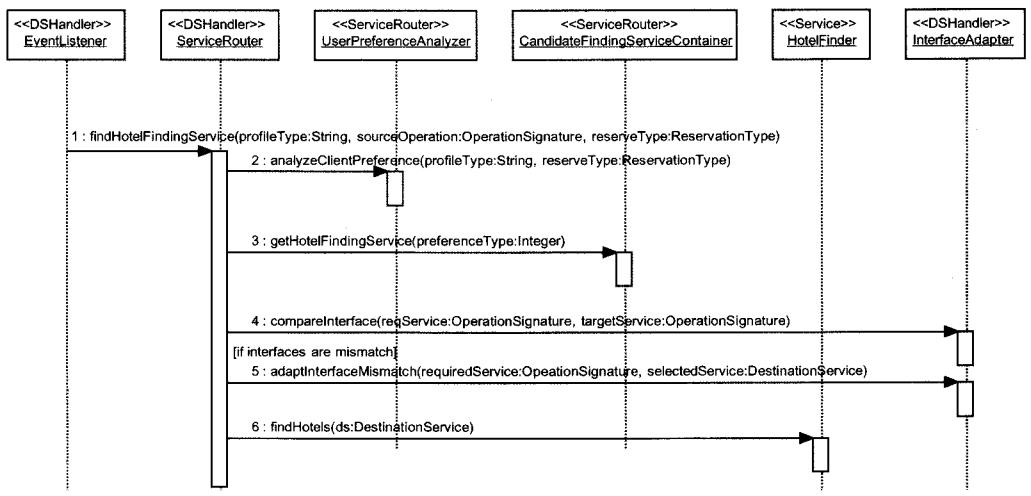

그림 17 Hotel Finding Service 시퀀스 다이어그램

표 1 제안된 동적 서비스 선택 기법과 기존 연구와의 비교(✓ : 제공)

연구 / 지원항목	Fujii	Penta	Yu	본논문의 기법
실시간 서비스 선택	✓	✓	✓	✓
시맨틱 정보 제공	✓	✓	✓	✓
다양한 서비스 자원 고려				✓
프로세스 재배포로 인한 오버헤드 고려		✓		✓
비가용 서비스 발견		✓		✓
개인화 지원	✓	✓		✓
서비스 적응 지원				✓

려하였다. 프로세스를 재배포함으로써 발생되는 오버헤드를 고려하여 프락시를 이용하여 적합한 서비스를 수행하였다. 가용하지 않는 서비스를 발견하고 조합하는 과정에서 서비스 인터페이스의 불일치에 관한 고려가 미흡하였다.

Yu의 연구는 서비스 선택을 쉽게 하기 위한 아키텍처를 설계하였다. QoS 요구사항에 부합한 서비스 조합을 위한 여러 알고리즘을 설명하였고 각 알고리즘에서 발생되는 문제를 해결하기 위한 향상된 알고리즘을 제안하였다. 가용하지 않는 서비스에 대한 발견과 서비스 클라이언트의 선호도에 관한 지원이 부족하였다.

본 연구에서 제안한 기법은 동적 서비스 선택을 위한 여러 가지 특성을 기반한 설계 기법을 제안하였다. 일반적으로 널리 받아들여지고 있는 설계 패턴을 적용함으로써 재사용성을 높이고, 프로세스에 정의된 추상 서비스 컴포넌트와 동적으로 발견된 서비스 간의 불일치를 해결하기 위해 서비스 적응을 고려하였다. 다양한 환경에 배치되어 있는 서비스를 지원하기 위하여 관리 계층인 ESB를 이용하여 실현 가능성을 보였다.

7. 결 론

SOC에서 동적 서비스 선택은 서비스 클라이언트 측의 어플리케이션을 변경하지 않고 새로 발행되거나 변경된 서비스를 동적으로 발견하여 서비스 선택을 만드는 것을 가능하게 한다. 이는 SOC의 유일한 특징 중 하나로서, 한정된 서비스를 이용하여 다양한 서비스 클라이언트와 서비스 클라이언트의 다양한 요청에 맞는 서비스들을 유연하게 조합시킬 수 있고, 실시간에 서비스를 적응시킬 수 있는 이점을 가지고 있다.

본 논문은 많이 사용되는 버스 아키텍처인 ESB을 이용하여 동적 서비스 선택을 위한 기법을 제안하였다. 먼저, 동적 서비스 선택에 관련된 기존 연구와 ESB에 대해 간략히 정리하였고, SOC에서 동적 서비스 선택의 의미를 정확히 규명하기 위해 기존 관련 연구에서 정의한 동적 서비스 선택의 의미를 분석하였다. 그리고, 동적 서비스 선택을 위한 설계 기법인 동적 선택 핸들러(Dynamic Selection Handler, DSH)를 제안하였다. DSH는 Invocation Listener, Service Selector, Service Binder, Interface Transformer로 이루어져 있으며, 각 컴포넌트는 적절한 디자인 패턴을 이용하여 설계되었다. 이 DSH를 이용하여 서비스는 동적으로 발견되고, 서비스 요청에 맞게 선택되고 적응될 수 있다. 그리고 제안된 동적 서비스 선택 기법을 ESB 플랫폼에서 구현하였다.

본 논문에서 제안된 기법을 이용하여, 동적 서비스 선택으로 인한 다양한 이점을 얻을 수 있었다. 우선, 비즈니스 프로세스를 생성할 때 해당 서비스 컴포넌트의 교체로 인한 비즈니스 프로세스의 재 배포를 막기 위하여 웹 서비스 인터페이스를 명세 수준의 인터페이스(BPEL에 명시된 WSDL 인터페이스)와 구현 수준의 인터페이스(웹 서비스의 WSDL 인터페이스)로 분류하였다. DSH는 분리된 두 개의 인터페이스 정보를 이용하여 동적 서비스 선택을 관리하였다. 일반적으로 널리 받아들여지고 있는 디자인 패턴 (Delegation 패턴 및 Adapter 패턴)을 DSH 설계에 적용하여 확장 가능성에 대한 고려를 하였다. 그리하여 특정 비즈니스 프로세스와 웹 서비스와 독립적으로 동적 서비스 선택을 유연하게 만들 수 있었다. 둘째 리스너는 실시간에 동적 서비스 선택의 요청을 인식할 수 있기 때문에, 실시간에 클라이언트 선호도 또는 프로파일 정보에 적절한 웹 서비스 컴포넌트를 결정하고 이들을 조합할 수 있게 되었다. 마지막으로 DSH는 서비스 통합을 위해 많이 사용되는 ESB 플랫폼을 기반으로 구현하였기 때문에, 본 논문에서 제안한 기법은 SOC에서 동적 서비스 선택의 실현 가능성을 보여주었다.

DSH를 구성하는 모든 컴포넌트에 대한 구현은 계속 진행 중에 있으며, 서비스 클라이언트의 요청과 프로파일 정보에 맞는 정확한 서비스들을 선택하고 조합하기 위해 클라이언트 선호도를 분석하는 기법을 개발할 것이다. 그리고, 어플리케이션에 종속적인 QoS를 기반으로 동적 서비스 선택을 효과적으로 할 수 있는 기법에 대한 연구도 진행 중에 있다.

참 고 문 헌

[1] Fujii, K. and Suda, T., "Semantics-Based Dynamic Service Composition," IEEE Journal on Selected Areas in Communications, Vol.23, No.12, pp. 2361-2372, 2005.

[2] Penta, M., Esposito, R., Villani, M., Codato, R., Colombo, M. and Nitto, E., "WS Binder: a framework to enable dynamic binding of composite web services," On the proceedings of the 2006 inter-

national workshop on Service-oriented software engineering (IW-SOSE'06), ACM, 2006.
[3] Yu. T., Zhang, Y., and Lin, K.J., "Efficient Algorithms for web Services Selection with End-to-End QoS Constraints," ACM Transactions on the Web, ACM, Vol.1, No.1, 2007.
[4] Denaro, G. Pesse, M., Tosi, D., and Schilling, D., "Toward Self-Adaptive Service-Oriented Architectures," On the proceedings of the Workshop on Testing, Analysis and Verification of Web Services and Applications (TAV-WEB '06), ACM, 2006.
[5] Zahreddine, W., and Mahmoud, Q., "A Framework for Automatic and Dynamic Composition of Personalized Web Services," On the proceedings of the 19th international conference on Advanced Information Networking and Application (AINA '05), IEEE, 2005.
[6] Benatallah, B., Casati, F., Grigori, D., Nezhad, H., and toumani, F., "Developing Adapters for Web Services Integration," On the proceedings of The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), LNCS 3520, Springer, 2005.
[7] Chappell, D.A., Enterprise Service Bus, O'Reilly, 2004.
[8] Java Business Integration (JBI), http://java.sun.com/integration/1.0/docs/sdk/api/index.html.
[9] The OWL Services Coalition, "OWL-S: Semantic Markup for Web Services," http://www.daml.org/services/.
[10] Majithia, S., Walker, W., D., Gray, A., W., "Automated Web Service Composition Using Semantic Web Technologies," First International Conference on Autonomic Computing (ICAC'04), pp. 306-307, 2004.
[11] Celtix, http://celtix.objectweb.org/
[12] ServiceMix, http://www.servicemix.org/
[13] Open ESB, https://oepn-esb.dev.java.net/

배 정 섭
2005년 전남대학교 물리학과 이학사. 2005년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 서비스 지향 아키텍처(SOA), 비즈니스 프로세스 모델링(BPM), 컴포넌트 기반 개발(CBD)

라 현 정
2003년 경희대학교 우주과학과 이학사 2006년 숭실대학교 컴퓨터학과 공학석사 2006년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 서비스 지향 아키텍처(SOA), 컴포넌트 기반 개발(CBD)

김 수 동
1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 교수. 관심분야는 서비스 지향 아키텍처(SOA), 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 소프트웨어 아키텍처