http://dx.doi.org/10.3745/KIPSTD.2011.18D.6.461

컴포넌트 기반 개발 (CBD) 설계의 서비스 지향 아키텍처 (SOA) 설계로의 실용적인 변환 기법

천 두 완[†] · 조 성 현[††] · 김 수 동[†††]

요 약

컴포넌트 기반 개발(CBD)는 재사용 컴포넌트를 개발하고 이를 결합하여 타겟 애플리케이션을 개발하는데 초점을 맞추고 있다. 서비스 지향 컴퓨팅(SOC)는 재사용 가능한 서비스를 개발하고, 발행하고, 조합하여 타겟 서비스 애플리케이션을 개발하는 비교적 새로운 패러다임이다. CBD의 대중성으로 인해, 많은 기관에서 이미 그들의 애플리케이션을 위한 CBD 모델을 소유하고 있다. 그러므로 기존에 있는 CBD 모델을 변환하여 SOC 애플리케이션을 개발하는 것이 처음부터 SOC 애플리케이션을 개발하는 것보다 경제적이다. 본 논문에서 기존에 많은 프로젝트를 통하여 보유하고 있는 CBD 설계를 SOA 설계로 변환하는 체계적인 방법을 다룸으로써 SOA 프로젝트의 실패율을 줄이고, 경제성을 확보한다. 이를 위해 먼저 CBD 설계와 SOA 설계의 구성 요소를 정리하고, 비교한다. 비교된 결과를 기반으로 CBD 설계와 SOA 설계의 매핑 관계를 정의하고, 이를 기반으로 변환 기법을 제안한다. 이 변환 기법 응용의 장점은 기존 CBD 설계와 SOA 설계 간의 일관성을 유지하면서 기존 설계의 재사용을 가능하게 하고, 서비스 지향 애플리케이션을 개발하는데 비용을 절감하는 데에 있다.

키워드 : CBD 설계, SOA 설계, 변환, 플랫폼 독립적인 설계, 모델 간 변환

A Practical Method to Transform Component-based Design to Service-Oriented Design

Du Wan Cheun[†] · Sung Hyun Jo[††] · Soo Dong Kim[†††]

ABSTRACT

Component-Based Development (CBD) focuses on developing reusable components and assembling them into target applications. Service-Oriented Computing (SOC) is a relatively new paradigm where reusable services are developed, published, and composed into target service applications. Due to the popularity of CBD, many organizations already have CBD models for their applications. Hence, it is feasible and appealing to develop SOC applications economically by transforming existing CBD models, rather than developing SOC applications from the scratch. Our research is to develop a methodology for transforming existing CBD design into Service-Oriented Architecture (SOA) design. In this paper, we first compare the two paradigms and their key elements for the design; CBD design and SOA design. Then, we define mapping relationships of the transformation. For each transformation mapping, we present step-wise instructions. The benefit of applying this transformation is to increase reusability of existing design assets, to maintain the consistency between existing CBD and SOC models, and reduced cost for deploying service-oriented applications.

Keywords : CBD Design, SOA Design, Transformation, Platform Independent Design, Model-to-model Transformation

1. 서 론

소프트웨어 재사용은 소프트웨어 위기를 해결하기 위해 나온 방법 중 하나로써 소프트웨어 개발 시 기존에 있는 소프트웨어를 활용하여 원하는 소프트웨어를 만들고자 하는 것이다[1]. 즉, 소프트웨어 개발 방법은 재사용 패러다임을 기반으로 발전하고 있다. 이 중 컴포넌트 기반 소프트웨어 개발 방법 (Component-based Software Development, CBD)은 업계에서 널리 사용하고 있는 재사용 패러다임 중

※ 이 논문은 정보통신산업진흥원의 SW공학 요소기술 연구개발사업에 의해 지원되었음.
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 연구되었음(No.2009-0076392).
† 준 회 원 : 숭실대학교 컴퓨터학과 박사수료
†† 정 회 원 : 정보통신산업진흥원 소프트웨어공학연구팀 책임연구원
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2011년 5월 23일
수 정 일 : 1차 2011년 6월 24일
심사완료 : 2011년 7월 12일

하나이다[2]. 이 방법에서는 컴포넌트를 재사용할 수 있도록 개발하고, 이런 컴포넌트를 통합하고 필요한 추가 기능을 구현함으로써 원하는 애플리케이션을 개발한다.

컴포넌트 기반 소프트웨어 개발 방법 이후에 소개된 서비스 지향 아키텍처(Service-Oriented Architecture, SOA) 기반 소프트웨어 개발 방법은 컴포넌트보다 더 높은 추상 레벨의 요소를 재사용하는 방법으로 많은 업계에서 이 패러다임을 활용하기 위해서 노력하고 있다[3]. 이 방법에서는 먼저 서비스를 재사용할 수 있도록 개발하고, 개발된 서비스를 재사용할 수 있도록 표준 인터페이스를 명세한다. 그런 다음, 서비스와 서비스 인터페이스를 발행하여 공개하여서, 여러 사람들이 사용할 수 있게 한다. 즉, 서비스 지향 소프트웨어 개발 방법은 재사용 가능한 서비스를 조합하여 서비스 기반 애플리케이션을 개발하는 것이다. 이처럼 재사용 가능한 서비스를 발행하고, 발행된 서비스를 재사용하여 원하는 기능을 사용하는 것을 서비스 지향 컴퓨팅(Service-oriented Computing, SOC) 이라고 하는데, 이를 가능하게 하기 위하여 OASIS, W3C와 같은 단체에서 SOAP, UDDI, WSDL, BPEL과 같은 표준을 정의하고, 이런 표준에 따라 서비스를 개발할 것을 권고하고 있다. 그러나 많은 기업에서 서비스 지향 컴퓨팅을 활용한 프로젝트를 수행함에 있어서 어려움을 겪었으며, 많은 수의 프로젝트가 실패로 끝나는 경우도 많이 존재하였다[4]. 그 이유는 서비스 지향 컴퓨팅에 대한 이해 부족과 체계적인 방법론 부족, 기존 시스템의 충분하지 않은 활용 등이었다.

우리는 이를 해결하기 위하여 대표적인 CBD 방법론인 마르미 III를 비롯한 다른 CBD 방법론을 분석하고[5][6], SOA 관련된 표준 및 참조 모델 등을 분석하였다[7][8]. 그 결과 SOC를 위한 대부분의 표준이 CBD 개념과 구조를 기반으로 하고 있음을 알 수 있었다. 또한, 이미 많은 조직이 CBD 기반으로 프로젝트를 수행하고, 애플리케이션을 개발하였으므로 재사용할 수 있는 CBD 모델을 소유하고 있음을 알 수 있었다. 그러므로 SOD 모델을 개발하고 서비스 기반 애플리케이션을 개발함에 있어서, 기존의 CBD 모델을 변환하여 개발하는 것이 처음부터 개발하는 것보다 경제적이며, 소프트웨어 재사용 패러다임에도 부합한다.

그러므로 본 연구에서는 CBD 설계를 SOA 설계로 변환하는 체계적인 방법에 초점을 맞추고 있다. 이를 위하여 먼저 CBD 설계와 SOA 설계의 구성 요소를 정리하고, 비교한다. SOA 설계 모델이 CBD 설계 모델을 기반으로 하고 있으나, SOC를 위한 추가적인 장치가 SOA 설계에 포함되어 있으므로, CBD 설계를 기반으로 변환할 수 있는 범위를 정의한다. 그런 다음, CBD 설계를 구성하는 요소를 SOA 설계로 변환하는데 필요한 기법을 정의한다. 각 기법을 정의함에 있어 단계별 지시사항 및 예제를 제공함으로써, 변환 기법의 효용성을 향상 시킨다. 또한, 사례 연구를 통하여 각 기법의 적용 순서 및 실효성을 보여준다. 마지막으로, CBD 설계를 변환하여 얻을 수 있는 SOA 설계와 추가적으로 설계하여야 하는 SOA 모델을 정리함으로써, CBD 설계 변환의 범위를 정의한다. 제안한 변환 기법을 통해 효과적으로 기존에 있는 CBD 설계를 SOA 설계로 변환하고, 서비스 기반 애플리케이션을 구현할 수 있을 것이다. 이 변환 기법 응용의 장점은 기존 CBD 설계와 SOA 설계 간의 일관성을 유지하면서 기존 설계의 재사용을 가능하게 하고, 서비스 지향 애플리케이션을 개발하는데 비용을 절감하는 데에 있다.

2. 관련 연구

Lee의 연구는 기존에 있는 CBD 워크벤치를 활용하여 웹 서비스 기반의 애플리케이션을 구현하는 방법을 제안한다[9]. 이를 위하여, 이 연구에서는 구현된 컴포넌트의 기능을 활용하여 웹 서비스로 래핑하는 방법을 사용하고 있다. 즉, 기존에 있는 컴포넌트 구현을 바꾸지 않고, 외부에 추가적인 장치를 구현함으로써 CBD 기반의 애플리케이션을 서비스 기반 애플리케이션으로 변환하는 데에 초점을 맞추고 있다. 그러나 컴포넌트의 배포형태는 바이너리 코드로써 기능성을 제공하는 데에는 충분하지만, 컴포넌트 자체가 서비스로 일대 일 매핑이 될 수도 있고, 그렇지 않을 수도 있다. 이를 해결하기 위해서는 CBD 설계를 활용하여야 하지만, Lee의 연구에서는 CBD 구현 결과를 활용하고 있기 때문에, SOA에서 말하는 서비스 개념을 만족하는 서비스를 설계 및 구현하는 데에는 한계점이 있다.

Brown의 연구는 본 연구와 비슷하게 컴포넌트에서 웹 서비스의 변환에 대해 다룬다[10]. 이를 위해 먼저 컴포넌트와 웹 서비스 간의 공통성 및 가변성을 분석한다. 이를 기반으로 컴포넌트에서 웹 서비스의 변환을 위한 주요 활동을 정의한다. 주요 활동은 크게 두 가지로 정제(Refinement)와 변환(Transformation)이 그것이다. 정제 활동은 컴포넌트에서 서비스로 정제하기 위해 필요한 비즈니스적인 관점에 대해 다루고 있고, 변환 활동은 정제 활동에서 나온 결과물을 기반으로 한 리팩토링, 모델 간 변환, 모델-코드 변환을 통하여 웹 서비스를 도출하고 있다. 그러나 Brown의 연구에서 모델 간 변환 및 모델-코드 변환을 다루고 있으나, CBD 설계와 SOA 설계의 주요 요소를 활용한 측면이 아니라 컴포넌트와 서비스의 비즈니스적인 관점의 차이를 기반으로 변환을 기술하고 있다. 즉, 실제 CBD 설계를 SOA 설계로 변환하는 데에는 한계점이 있다.

Jiang의 연구는 컴포넌트 기반으로 서비스를 만드는 프레임워크를 제안하고 있다[11]. 이 프레임워크를 통해서 모바일 데이터 서비스의 설계와 개발을 단순화하는데 목적을 두고 있다. 즉, 이 프레임워크를 사용하여, 모바일 서비스의 구조, 행위, 배치 등을 UML로 설계할 수 있는 기반을 제공하고 있다. 그러나 이 연구에서는 서비스를 설계하는 데에 있어서 CBD 설계 기술이 어떻게 사용될 수 있는 지에 초점을 맞추고 있어서, 기존에 우리가 갖고 있는 CBD 설계 자산을 활용하여 SOA 설계를 도출하는 데에는 한계가 있다.

현재, 대부분의 연구에서 서비스를 설계하고 개발하려면 컴포넌트의 역할이 중요하다는 것을 역설하고 있다. 그러나 컴포넌트 구현 결과를 활용하거나, 비즈니스 관점에서의 CBD

설계와 SOA 설계의 변환을 다룸으로써, 변환의 추상화 레벨이 너무 높거나, 너무 낮다. 즉, 이미 우리가 갖고 있을 수 있는 CBD 설계를 SOA 설계로 변환하여 기존 설계의 재사용을 가능하게 하고, 비용을 절감하게 하기에는 한계점이 있다.

3. CBD 설계와 SOA 설계의 메타 모델

3.1 CBD 설계의 핵심 요소

CBD 설계는 크게 컴포넌트 외부 상호 작용을 파악하는 것과 내부의 구조를 파악하는 것으로 나뉜다. 따라서 마르미 III과 같은 CBD 관련 연구에서는 컴포넌트 외부 상호 작용을 파악하기 위한 단계와 내부의 구조를 파악하기 위한 단계를 정의하고 있으며, 각 단계마다 다른 종류의 산출물이 존재한다. 우리는 이런 관련 연구에서 정의하고 있는 단계 및 산출물에 따라 (그림 1)과 같이 CBD 설계의 메타모델을 유도하였다.

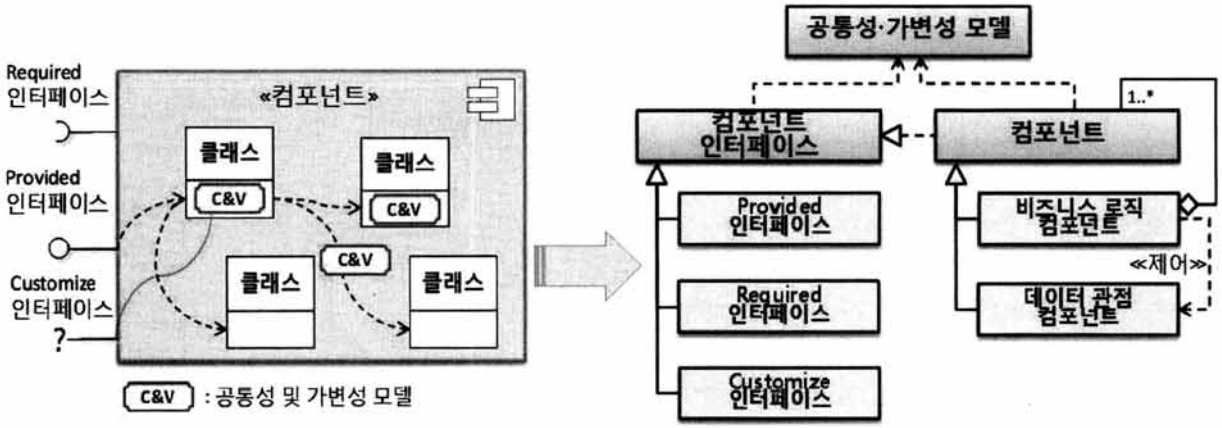
컴포넌트: 컴포넌트는 크게 비즈니스 컴포넌트와 응용 컴포넌트로 나눌 수 있다[5][6]. 비즈니스 컴포넌트란 구현의 대상이 되는 데이터 관점의 컴포넌트와 비즈니스 로직을 포함하는 컴포넌트를 말한다. 응용 컴포넌트란 비즈니스 컴포넌트의 구현을 위한 기술적인 컴포넌트를 말하는 것으로서 대부분 응용 컴포넌트는 프레임워크(Framework) 기술을 이용하여 구현하게 된다. 또한 응용 컴포넌트는 기술 아키텍처 위에서 비즈니스 컴포넌트의 수행을 지원하는 역할을 수행한다. 재사용성이 높은 비즈니스 컴포넌트를 개발하기 위해서는 범용성이 있으면서 확장가능하고 운영체제와 독립적인(Platform Independent) 응용 아키텍처 위에서 구현해야 한다. 본 논문의 메타모델에서 '컴포넌트'는 비즈니스 컴포넌트로써 CBD에서 기능을 수행하는 독립적인 모듈의 기본적인 단위이다. 이런 컴포넌트는 하나 이상의 클래스를 포함하고 있다. 또한, 컴포넌트는 하나 이상의 다른 컴포넌트를 포함하여 복합 컴포넌트를 구성할 수 있다. 그러므로 컴포넌트는 객체와 비교하여 재사용 단위가 크다.

비즈니스 컴포넌트는 일반적으로 비즈니스 로직 컴포넌트와 데이터 관점 컴포넌트로 나뉜다. 비즈니스 로직 컴포넌트는 비즈니스 과정 내의 직무나, 절차서 내의 작업 단계들, 단계 별 요구되는 입력 및 출력 정보를 명세한다. 이런 작업 단계들은 워크플로우로 나타난다. 즉, 워크플로우에는 서로 독립적인 작업을 수행하기 위한 컴포넌트의 실행 순서 및 실행 패턴을 명세한다. 이런 비즈니스 로직 컴포넌트를 설계하기 위해서는 UML (Unified Modeling Language)의 액티비티 다이어그램, 시퀀스 다이어그램이나 커뮤니케이션 다이어그램 등을 활용할 수 있다. 데이터 관점 컴포넌트는 하나 이상의 엔티티 클래스로 구성된다. 엔티티 클래스는 일반적으로 컴포넌트에서 기능 수행을 하기 위하여 필요한 데이터 처리를 위한 함수를 제공한다. 즉, 데이터 관점 컴포넌트는 지속적으로 데이터 관리나, 비즈니스 로직 컴포넌트에서 요구하는 질의에 따른 데이터 처리를 위한 함수를 제공한다. 이런 데이터 관점 컴포넌트를 설계하기 위해서는 UML의 클래스 다이어그램이나 객체 다이어그램 등을 활용할 수 있다.

컴포넌트 인터페이스: CBD에서 인터페이스는 컴포넌트와 분리되어 구현한다. 예를 들어 엔터프라이즈 자바 빈즈 (Enterprise Java Beans)나 코바 (CORBA)와 같은 컴포넌트 기반 프로그래밍 언어에서 컴포넌트와 인터페이스를 구현하기 위한 구성 요소를 각각 제공하고 있다.

컴포넌트 인터페이스는 일반적으로 인터페이스의 역할에 따라 Provided 인터페이스, Required 인터페이스, Customize 인터페이스로 나뉜다[12]. Provided 인터페이스는 컴포넌트가 제공하는 기능을 명세하기 위하여 사용한다. 예를 들어 컴포넌트 A가 컴포넌트 B의 기능을 사용하는 경우를 고려해보자. 이런 경우 컴포넌트 A는 컴포넌트 B의 Provided 인터페이스를 사용하여, 관련 기능을 사용하게 된다.

Required 인터페이스는 하나의 컴포넌트가 관련 기능을 수행함에 있어서 필요로 하는 기능을 명세하기 위하여 사용한다. 예를 들어, 컴포넌트 A가 기능을 수행할 때 관련 하위 기능이 여러 개 있다고 하자. 그렇지만 모든 기능을 컴포넌트 A에서 제공할 수 없기 때문에, 일부 기능은 다른 컴포넌트의 기능을 활용하여야 한다. 즉, 컴포넌트 A에서 제공하지 않고, 다른 컴포넌트에서 제공하는 기능을 활용하기 위하여, 관련 기능을 묶어서 인터페이스화하고 이를 명세한



(그림 1) CBD 설계의 메타모델

것이 컴포넌트 A의 Required 인터페이스가 된다. 결과적으로 Required 인터페이스와 Provided 인터페이스가 부합하는 것을 찾아서 컴포넌트 간의 조합을 수행하게 된다.

Customize 인터페이스는 컴포넌트가 제공하는 기능성을 사용자의 요구에 맞게 특화할 수 있는 부분을 명세하기 위하여 사용한다. 컴포넌트는 재사용을 위하여 설계되고 개발된 것이기 때문에, 일반적으로 여러 애플리케이션에서 사용될 수 있도록 구성된다. 이런 컴포넌트의 재사용성을 향상시키기 위하여 사용자의 요구에 맞게 특화시킬 수 있는 부분이 내부적으로 설계 및 구현되어 있으며, 이를 외부에서 가능하게 해주는 것이 Customize 인터페이스이다.

이런 세 가지 인터페이스를 명세한 문서를 컴포넌트 인터페이스 명세서라 하며, 명세서에는 인터페이스가 포함하고 있는 메소드 이름, 각 메소드의 매개 변수 및 반환 자료형이 기술된다. 또한, 인터페이스의 사용상의 의미를 기술하기 위하여, 각 메소드 실행 시 고려하여야 할 사전 조건, 사후 조건 및 제약 사항을 기술한다.

공통성 및 가변성(Commonality and Variability, C&V) 모델: 컴포넌트는 여러 애플리케이션에서 재사용할 수 있도록, 애플리케이션들의 공통성 및 가변성이 설계되어 구현된다[12]. 이런 공통성 및 가변성은 컴포넌트 인터페이스 및 컴포넌트 구현에 영향을 끼친다. 즉, 공통성은 컴포넌트의 기능성이 되고, 가변성은 이런 공통 기능성 중 특화할 수 있는 부분과 매핑이 되며, 이런 특화를 외부에서 가능하게 하기 위하여 커스터마이즈 인터페이스를 제공한다. 그러므로 공통성과 가변성은 설계되고 문서화되어야 한다. 그러므로 공통성 및 가변성 모델은 공통적인 특성과 공통적인 특성 중 가변적인 특성을 명세하고 있다. 또한, 가변적인 특성을 명세하기 위하여 가변이 발생하는 가변점과 가변점에 할당(binding)할 수 있는 유효한 가변치를 명세한다.

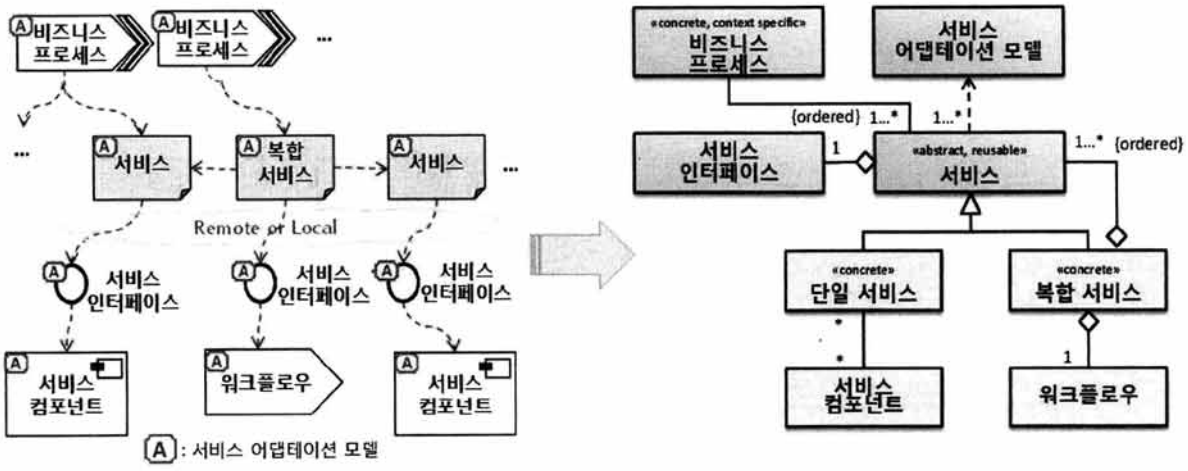
3.2 SOA 설계의 핵심 요소

SOA 기반 애플리케이션은 일반적으로 계층적인 아키텍처를 기반으로 하고 있다. 우리는 SOC의 핵심 산출물을 정의하고 이를 기반으로 서비스 지향 설계에서 주요한 핵심 요소를 도출한다[7][8][13]. (그림 2)는 이런 핵심요소와 요소 간의 관계를 정의한 메타 모델을 보여준다.

비즈니스 프로세스(Business Process, BP): 비즈니스 프로세스는 서비스 소비자 관점에서 정의된 작업의 순서를 정의한 것으로, 사용자가 기업에서 수행하는 비즈니스 활동 또는 실행기술을 나타낸다. 비즈니스 프로세스는 비즈니스의 주요한 기능을 기술한다. 하나의 비즈니스 프로세스는 여러 개의 하위 프로세스로 분해될 수 있고, 하위 프로세스는 더 작은 단위로 분해될 수 있다. 일반적으로 BP는 유즈케이스(use case)와 객체의 메소드보다 비교적 단위가 크고 작은 단위의 액티비티(activity) 간의 워크플로우로 정의된다. 현재 비즈니스 프로세스를 명세하는 데에는 BPEL이 가장 널리 사용되고 있다.

서비스: 서비스는 서비스 소비자가 원하는 요구사항을 만족시킬 수 있는 하나 이상의 기능을 포함하는 단위이다. 서비스 소비자는 자신이 필요로 하는 요구사항을 만족시킬 수 있는 서비스를 호출하고, 서비스 제공자는 수익을 창출할 수 있는 기능을 서비스 형태로 제공한다. 즉, 서비스는 소비자의 요구사항에 맞는 기능을 서비스 제공자가 제공할 수 있도록 하는 수단이다. 서비스는 제공되는 기능의 크기 또는 범위에 따라서 단일 서비스 혹은 복합 서비스로 정의될 수 있다.

단일 서비스(atomic service)는 더 이상 분해되지 않는 기능 단위로, 단순한 기능을 수행하는 태스크를 나타낸다. 단일 서비스는 필요한 기능을 모두 포함하고 있으므로, 다른 서비스와 상호작용하거나 다른 서비스에 의존하지 않고 독자적으로 기능을 수행한다. 이런 단일 서비스는 서비스 컴포넌트(service component)로 구현된다. 서비스는 서비스 인터페이스로만 서비스 소비자에게 노출되고, 서비스의 내부 구현을 숨겨져 있다. 그러므로 인터페이스가 잘 정의되어 있다면, 서비스 내부인 서비스 컴포넌트는 다양한 프로그래밍 패러다임을 이용하여 구현될 수 있다. 서비스 컴포넌트는 객체 지향 프로그래밍의 객체 또는 컴포넌트 기반 개발의 컴포넌트로 구현된다.



(그림 2) SOA 설계의 메타모델

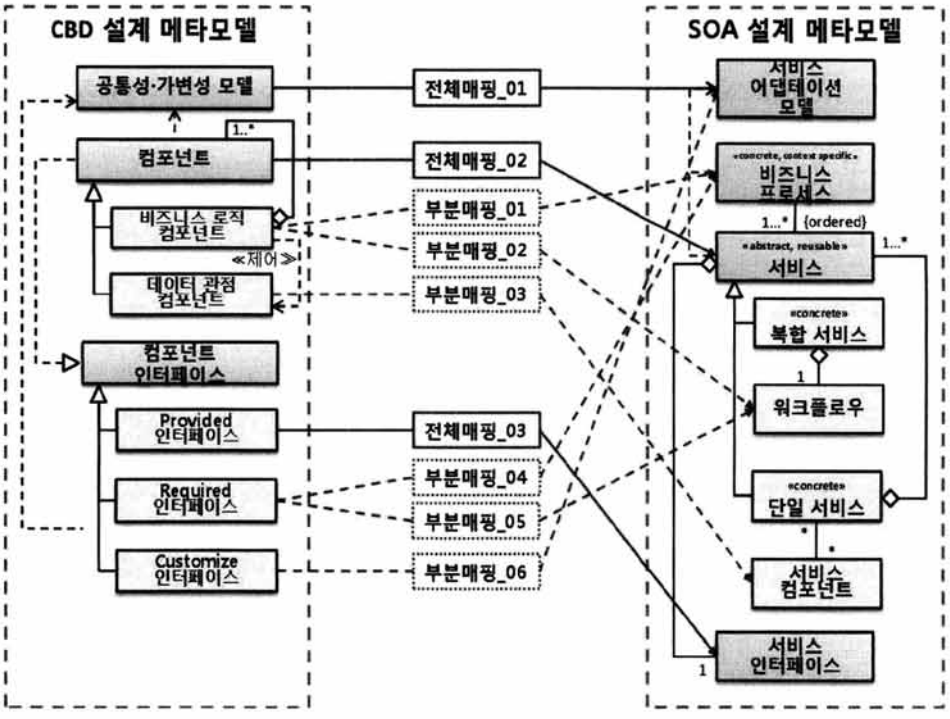
복합 서비스(composite service)는 하나 이상의 서비스로 구성되어 워크플로우(workflow)를 나타내며, 포함되는 서비스는 단일 서비스 또는 또 다른 복합 서비스가 될 수 있다. 워크플로우는 복합 서비스에 포함되는 여러 서비스간의 제어 흐름을 나타내며, 일반적으로 SOA에서는 워크플로우가 BPEL 문서로 기술된다. 비즈니스 프로세스의 워크플로우와 복합 서비스의 워크플로우는 명세하는 방법은 BPEL로 동일하다. 그러나 비즈니스 프로세스의 경우 특정 애플리케이션 하나 만을 위해 정의하지만, 복합 서비스의 경우 여러 애플리케이션에서 재사용할 수 있는 목적으로 정의한다는 점이 다르다.

서비스 인터페이스: 서비스가 제공하는 기능은 서비스의 구현 내용을 숨긴 상태에서 오직 서비스 인터페이스를 통해 사용자에게 제공된다. 즉, 서비스 인터페이스는 서비스 소비자가 서비스와 상호작용할 수 있는 수단을 제공한다. 서비스 인터페이스는 특정 프로토콜, 오퍼레이션, 입력 데이터 등 서비스를 호출하는데 필요한 정보를 포함한다. 서비스 인터페이스는 서비스 소비자나 다른 서비스에서 참조할 수 있는 형태로 표현되며, 서비스 기능을 사용하거나 서비스로부터 실행 결과를 받기 위해 서비스 소비자가 제공해야 하는 정보를 기술한다. SOA에서 서비스 인터페이스는 WSDL을 이용해서 명세되고 서비스 저장소에 등록되며, UDDI와 같은 서비스 발견 메커니즘을 통해서 발견된다.

서비스 어댑테이션(adaptation) 모델: SOC에서는 서비스의 재사용을 위해 서비스는 공통적인 특성으로 필수적으로 갖고 있어야 한다. 서비스의 공통적인 특성은 사용자나 문맥(context)에 따라 조금씩 달라질 수 있는데, 이를 가변성이라 일컫는다. 그러므로, 공통성과 가변성(C&V)는 이 산출물에 서술되어야 하며, 적절한 어댑테이션 메커니즘(mechanism)이 서비스 컴포넌트 설계 시 내제되어야 한다. 서비스 어댑테이션 모델은 가변점과 각 서비스에 맞는 관련 있는 가변점을 포함하고 있다.

4. CBD 설계와 SOA 설계의 매핑(mapping)

CBD의 컴포넌트와 SOA의 서비스는 재사용하기 위한 목적으로 설계되고 구현된다. 그로 인하여 CBD 설계와 SOA 설계에는 다음과 같은 공통점이 있다. 먼저, CBD 설계와 SOA 설계는 각각 개발 대상인 컴포넌트와 서비스의 재사용성을 확보하기 위한 단계 및 관련 산출물이 존재한다는 점이다. 또한, CBD와 SOA에서는 각각 컴포넌트와 서비스를 사용하기 위하여, 블랙 박스 형태의 인터페이스를 분리하여 제공한다. 반면, 컴포넌트와 서비스는 각각 배포 형태와 관련 표준 및 기술이 다르기 때문에 이에 해당하는 기술에 따른 제약사항 및 고려사항 등에서는 차이점이 있다. 그러므로 CBD 설계와 SOA 설계를 매핑함에 있어서는 두 가지 형태가 존재한다. 먼저 전체매핑의 경우는 CBD 설계가 모두 SOA 설계로 변환됨을 의미한다. 즉, CBD 설계 결과를 있는 그대로 SOA 설계에서 사용할 수 있는 경우를 의미한다. 부분매핑의 경우는 CBD 설계 결과를 이용하여 SOA 설계를 유도함을 의미한다. SOA 설계 시 필요한 정보가 CBD 설계 결과에 포함되어 있지만, 있는 그대로 사용할 수 있는 것이 아닌 경우에 해당한다. 즉, (그림 3)에서처럼 CBD 설계와 SOA 설계에서의 매핑관계에서는 3개의 전체매핑 및 6개의 부분매핑이 존재한다.



(그림 3) CBD 설계와 SOA 설계의 전체매핑 관계

전체매핑 01. 공통성 및 가변성 모델과 서비스 어댑테이션 모델: CBD 설계 메타 모델 중 공통성 및 가변성 모델은 컴포넌트의 재사용을 위한 설계 산출물이다. 공통성 및 가변성을 모델링하기 위하여, 컴포넌트가 재사용될 수 있는 도메인의 요구사항을 분석하여 그 중 공통 요구사항을 식별하고, 공통 요구사항 중 가변적인 요구사항을 식별하여 정리한다. 그리고 공통성 및 가변성 모델을 토대로 컴포넌트를 설계하여 사용자의 요구에 맞춰 특화할 수 있는 부분을 설계 및 구현한다.

서비스 어댑테이션 모델 역시 서비스의 재사용성을 위한 설계 산출물이다. 서비스를 재사용할 수 있도록 설계하기 위해서는 서비스 요구사항의 공통성 및 가변성이 식별되어야 한다. 즉, 컴포넌트를 위한 공통성 및 가변성 모델의 산출물은 서비스 어댑테이션 모델에서 모두 활용할 수 있다.

전체매핑 02. 컴포넌트와 서비스: 컴포넌트는 역할에 따라서 비즈니스 로직 컴포넌트 및 데이터 관점 컴포넌트로 분류할 수 있지만, 이런 분류에 속하지 않는 컴포넌트도 존재한다. 이런 경우는 일반적으로 컴포넌트 내부에 컨트롤러 기능 및 모델 기능을 모두 포함하고 있으며, 독립성 및 모듈성이 높다. 일반적으로 서비스 또한 컨트롤러 기능 및 모델 기능을 모두 포함한다. 그로 인하여 사용자가 특정 기능을 요청하였을 때, 모든 기능을 내부적으로 수행할 수 있도록 설계 및 구현된다. 그러므로 이런 경우 컴포넌트 설계의 전부가 서비스 설계에 모두 활용된다.

전체매핑 03. Provided 인터페이스와 서비스 인터페이스: 컴포넌트 인터페이스 중 Provided 인터페이스는 컴포넌트가 제공하는 기능 및 필요로 하는 기능을 명세한다. 서비스 인터페이스 역시 서비스의 기능성을 공개하기 위하여 서비스가 제공하는 기능성을 명세하기 위해서 사용된다. 그러므로 컴포넌트 인터페이스 중 Provided 인터페이스를 이용하여 서비스 인터페이스를 도출할 수 있다.

부분매핑 01. 비즈니스 로직 컴포넌트와 비즈니스 프로세스: CBD에서 비즈니스 로직 컴포넌트는 마크로 워크플로우를 포함하고 있다. 이 마크로 워크플로우에는 다른 비즈니스 로직 컴포넌트나 데이터 관점 컴포넌트의 호출 순서가 포함되어 있으며, 이 호출 순서는 여러 곳에서 재사용할 수 있는 형태로 제공된다. 이런 워크플로우 설계에는 UML의 액티비티 다이어그램 등이 활용된다. 비즈니스 프로세스 역시 다른 서비스를 제어하기 위한 일련의 제어 흐름 (Control flow)을 갖고 있다. 제어 흐름에 따라서 다른 서비스들의 호출을 결정한다. 그러므로 컴포넌트의 마크로 워크플로우를 기반으로 비즈니스 프로세스의 제어 흐름을 도출할 수 있다. 그러나 비즈니스 로직 컴포넌트의 워크플로우는 재사용을 기반으로 설계되었으나, 비즈니스 프로세스의 제어 흐름은 특정 문맥(context)에 맞게 설계되어 있다. 그러므로 특정문맥에 맞게 재사용 가능한 워크플로우를 특화시키고, 문맥 특화적인 설계를 추가하여야 한다.

부분매핑 02. 비즈니스 로직 컴포넌트와 복합 서비스 워크플로우: 비즈니스 로직 컴포넌트와 복합 서비스의 워크플로우는 목적 측면에서 동일하다. 즉, 재사용할 수 있는 제어 흐름을 포함하고 있기 때문에 부분매핑 01과 달리 특화시키는 과정은 필요하지 않다. 그러나 비즈니스 로직 컴포넌트의 워크플로우와 복합 서비스의 워크플로우의 설계 결과가 다르기 때문에 비즈니스 로직 컴포넌트 워크플로우 설계의 내용을 가져와서 복합 서비스의 워크플로우 설계에 활용하는 부분매핑이 이루어진다.

부분매핑 03. 데이터 관점 컴포넌트와 서비스 컴포넌트: 데이터 관점 컴포넌트는 엔티티 클래스를 포함하고 있으며, 지속적으로 데이터 관리나, 비즈니스 로직 컴포넌트에서 요구하는 질의에 따른 데이터 처리를 위한 함수를 제공한다. 이런 기능은 서비스에서도 역시 필요하다. 즉, 서비스에서 제공하는 대부분의 기능성은 사용자에게 입력을 받아서 그 입력에 대한 처리를 한다거나, 요청에 따른 응답을 수행한다. 이를 위해서는 서비스를 구현하고 있는 컴포넌트 중에서 데이터 관점 컴포넌트의 기능성을 포함하고 있는 것이 있어야 하므로 전체 매핑될 수 있다. 즉, 서비스 컴포넌트가 데이터 관점 컴포넌트를 포함한다 할 수 있다.

부분매핑 04. Required 인터페이스와 비즈니스 프로세스: 비즈니스 프로세스는 다른 서비스의 호출을 제어하고 있다. 이를 위하여 연결되는 두 개의 서비스 간의 입력 및 출력의 매핑 관계를 정의하여야 한다. 이 매핑 관계를 정의하기 위하여 비즈니스 프로세스가 중재자 역할을 하는데, 이 때 비즈니스 프로세스 측면에서 필요로 하는 오퍼레이션 및 관련 속성을 명세하여야 한다. 즉, 비즈니스 프로세스 측면에서 필요로 하는 오퍼레이션 및 관련 속성의 의미하는 바는 서비스를 실행하기 위하여 선행 서비스에서 수행되어야 하는 오퍼레이션 및 관련 속성을 의미하게 된다. 즉, 이런 정보를 명세하기 위하여 컴포넌트의 Required 인터페이스가 활용될 수 있다.

부분매핑 05. Required 인터페이스와 워크플로우: 복합 서비스의 워크플로우는 비즈니스 프로세스의 제어 흐름 중 재사용 가능하다는 점만 다르기 때문에, 활용될 수 있는 이유는 부분매핑 04에서 언급한 것과 동일하다.

부분매핑 06. Customize 인터페이스와 서비스 어댑테이션 모델: 컴포넌트 인터페이스 중 커스터마이즈 인터페이스의 경우, 현재 서비스 인터페이스를 명세하는 대표적인 표준인 WSDL에서 명세할 수 있는 방법을 제공하고 있지 않다. 반면, Customize 인터페이스에서 컴포넌트를 특화하기 위하여 제공하는 오퍼레이션 집합들은 컴포넌트의 가변성 지원 정도를 요약해서 보여준다. 이 정보는 서비스 어댑테이션을 수행하기 위한 명세에서 사용될 수 있다.

이렇게 정의된 매핑 관계를 기반으로 5장에서 변환 기법을 정의한다. 그러나 (그림 3)의 매핑 관계에서 보듯이 모든

SOA 설계 산출물이 CBD 설계 산출물로부터 유도되지는 않는다. 단일 서비스 및 복합 서비스를 정의하기 위한 구성 요소는 CBD 설계 산출물로부터 유도되지만, 직접적으로 단일 서비스 및 복합 서비스를 정의할 수는 없다. 그 이유는 컴포넌트에서 정의하는 재사용 단위와 서비스에서 정의하는 재사용 단위가 차이가 있을 수 있기 때문이다. 그러므로 본 논문에서는 CBD 설계 산출물에서 SOA 설계 산출물로 변환이 되지 않는 단일 서비스 및 복합 서비스 설계에 대한 지침을 다룬다.

5. 매핑에 따른 변환 기법

5.1 전체 매핑을 위한 변환 기법

전체매핑 01. 공통성 및 가변성 모델에서 서비스 어댑테이션 모델로의 변환: CBD 가변성 모델은 가변점, 가변성 유형, 가변치, 범위를 기술하고 있다. CBD에서 가변점과 가변치는 컴포넌트에 설계되고 커스터마이즈 인터페이스를 통해 해결된다. SOA에서 서비스 어댑테이션은 프로토콜(protocol)과 연결, 메시지 형태, 순차나 오퍼레이션(operation), 이와 같이 세 개의 형태로 나뉘어진다. (그림 4)는 CBD의 공통성 및 가변성 모델에서 서비스 어댑테이션 모델로의 변환을 보여준다.

먼저, 컴포넌트 인터페이스의 가변성은 서비스 인터페이스의 가변성으로 변환된다. 일반적으로 컴포넌트 인터페이스의 가변점은 인터페이스를 구성하는 오퍼레이션의 입, 출력 메시지 유형 (예를 들어 매개 변수 개수, 매개 변수 타입)이다. 서비스 인터페이스를 구성하는 오퍼레이션의 입, 출력 메시지 유형은 일반적으로 XML 스키마를 이용하여 정의하게 되어 있으며, 스키마를 변경시킴에 따라 특화활동이 가능하다. 즉, 가능한 가변치에 해당하는 서비스 오퍼레이션의 입, 출력 메시지의 유형을 XML 스키마에 정의하고 정의된 여러 XML 스키마 중 요청에 따라 인터페이스 가변성을 특화시키도록 한다. 그러므로 컴포넌트 인터페이스의 가변성은 서비스 인터페이스와 관련있는 입, 출력 메시지 유형을 위한 XML 스키마로 매핑될 수 있다.

둘째, 컴포넌트의 워크플로우 가변성은 컴포넌트 내에 있는 컨트롤 클래스가 엔티티 클래스를 호출하는 순서에서의 가변성을 의미한다. 서비스 역시 컴포넌트처럼 컨트롤러 클래스와 엔티티 클래스로 구성된다. 즉, 서비스 내에서의 워크플로우 가변성 역시 서비스 내에 있는 컨트롤 클래스가 엔티티 클래스를 호출하는 순서에서의 가변성이 되며, 이에 따라 컴포넌트를 위해 설계해놓은 워크플로우 가변성이 그대로 매핑된다.

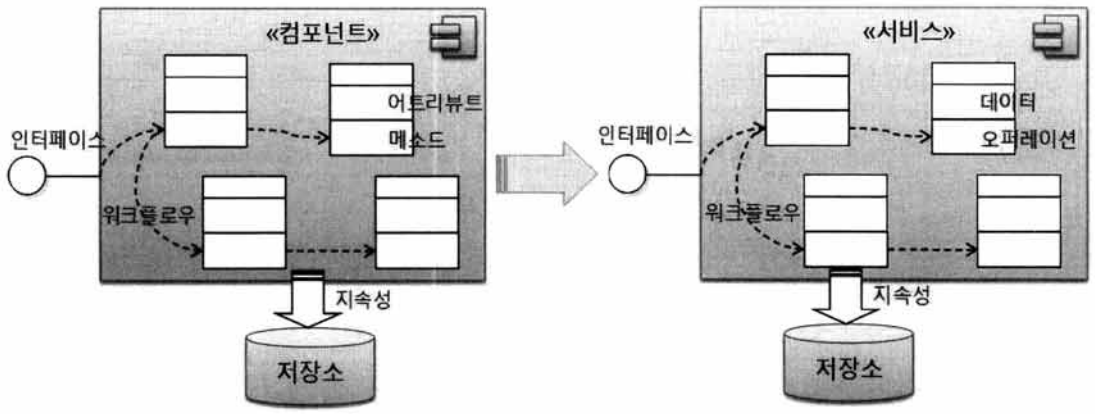
셋째, 컴포넌트의 어트리뷰트 가변성은 컴포넌트 내에 클래스를 구성하는 어트리뷰트의 개수 및 각어트리뷰트의 유형에서 가변성이 발생한다. 서비스를 구성하는 것 역시 클래스이므로 컴포넌트의 어트리뷰트 가변성을 모두 매핑할 수 있다.

넷째, 컴포넌트의 메소드 가변성은 컴포넌트를 구성하고 있는 클래스의 메소드를 구현하는 로직에서의 가변성을 의미한다. 서비스를 구성하는 것 역시 클래스이므로 컴포넌트의 메소드 가변성을 모두 매핑할 수 있다.

다섯째, 컴포넌트의 지속성 가변성은 컴포넌트에서 처리하는 데이터 중 외부 데이터베이스에 저장하는 부분에서의 가변성을 의미한다. 서비스 역시 내부에서 처리하는 데이터가 있으며 영구적으로 관리하기 위해서는 일반적으로 외부 데이터베이스를 사용한다. 즉, 컴포넌트에서 외부 데이터베이스를 사용하는 데 있어서 발생하는 가변성은 서비스에서 외부 데이터베이스를 사용하기 위해 필요한 가변성과 동일하다.

이를 위해서 세 가지 스텝을 적용할 수 있다. 첫째, 컴포넌트의 C&V 모델을 기반으로 서비스 컴포넌트의 공통성과 가변성을 식별한다. 둘째, 사용자가 서비스를 개인화할 수 있도록 문맥을 제공한다. 마지막으로, 서비스 어댑테이션 모델을 설계하고 명세한다.

전체매핑 02. 컴포넌트에서 서비스로의 변환: CBD 디자인에서는 전반적인 기능이 컴포넌트에 구조화되어 있다. 일반적으로 컴포넌트는 비즈니스 로직을 담당하는 비즈니스 로직 컴포넌트, 영구적인 데이터를 관리하는 데이터 컴포넌트로 나뉜다. 반면에, SOA 디자인에서는 전반적인 기능이 서비스에 구조화되어 있다. 컴포넌트는 설계 및 구현 단계



(그림 4) CBD의 C&V 모델과 서비스 어댑테이션 모델

의 유닛이지만, 서비스는 사용자가 인지할 수 있는 작업의 유닛이다. 이런 다름에도 불구하고, CBD의 컴포넌트 명세는 서비스 설명을 유도할 수 있는 주요 소스(source)이다.. 그 이유는 컴포넌트 명세와 서비스 설명 모두 타겟 애플리케이션의 전반적인 기능을 표현하고 있기 때문이다.

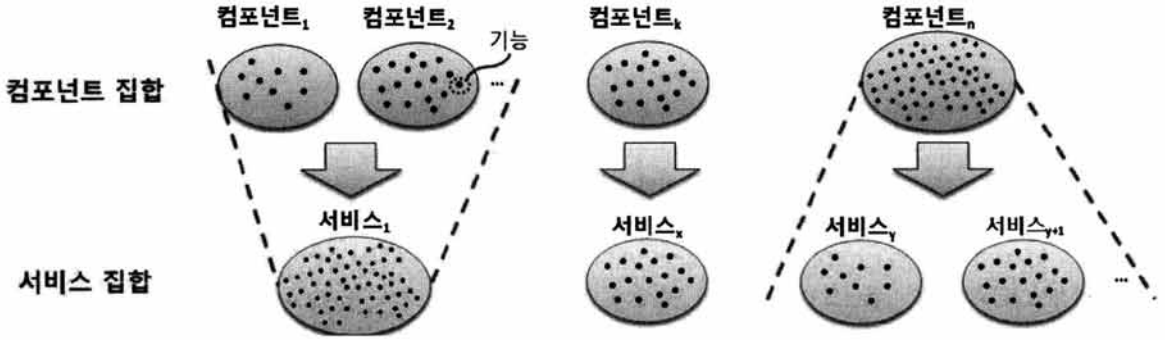
컴포넌트는 응집력이 좋은 독립적인 소프트웨어 모델이고, 서비스 또한 사용자에게 전달되는 작업의 독립적인 유닛이다. 그러나 컴포넌트와 서비스의 유닛 특성 상 언제나 일대 일 관계는 아니다. 그러므로 컴포넌트 명세로부터 우리가 기능, 시맨틱, 제약사항을 분석하고 이를 기반으로 컴포넌트 기능군을 분류한다. 그런 다음, 컴포넌트와 서비스 간 매핑 관계에서의 다중성(multiplicity)을 고려하여야 한다. (그림 5)에서처럼 컴포넌트와 서비스는 세 가지 유형의 매핑(일대 일, 다대 일, 일대 다)이 있다. 알맞은 매핑 형태를 결정하려면, 컴포넌트의 범위, 특성 등을 고려하여 분석하고, 이를 기반으로 정의해야 한다.

둘째, 컴포넌트의 유형별로 서비스에 매핑하여야 한다. 먼저, 비즈니스 로직 컴포넌트의 경우, 다른 컴포넌트의 호출을 제어하고 있으며, 이를 위한 워크플로우를 정의하고 있다. 또한 복합 서비스 역시, 다른 서비스의 호출을 제어하고 있으며 이를 위한 제어 흐름을 포함하고 있다. 그러므로 워크플로우를 제어 흐름으로 변환하는 방법을 고려하여야 한다. 여기서 흐름의 변환은 여러 가지 형태가 있으며, 자세한 사항은 컴포넌트에서 비즈니스 프로세스로의 변환에서 다룬다.

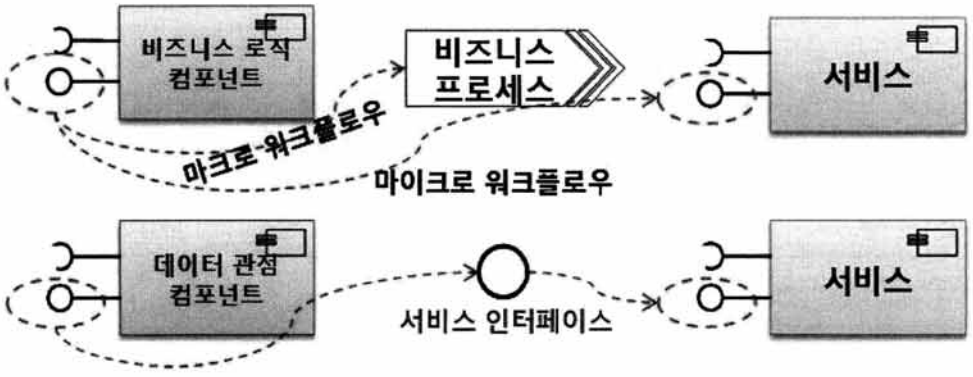
셋째, 컴포넌트와 서비스가 매핑된 이후에는, 컴포넌트의 오퍼레이션 집합과 서비스의 오퍼레이션 집합을 매핑하여야 한다. 서비스의 오퍼레이션 중 사용자가 사용할 수 있게 공개되는 오퍼레이션이 있고, 내부적으로만 사용하는 오퍼레이션이 있다. 즉, 컴포넌트에서 제공 인터페이스로 공개되는 오퍼레이션 집합을 기반으로 서비스 인터페이스로 공개되는 오퍼레이션으로 매핑이 이루어진다. 자세한 내용은 5장 2절에서 다룬다.

컴포넌트에서 서비스로의 변환을 위한 단계는 다음과 같다. 먼저, 컴포넌트 명세로부터 컴포넌트에서 제공하는 기능성을 도출한다. 그런 다음 컴포넌트의 기능 복잡도 및 재사용성을 확인하여 매핑 관계에서의 다중성을 고려한다. 그런 다음 컴포넌트 유형에 따라 서비스로 매핑하는데, 비즈니스 로직 컴포넌트의 경우 워크플로우 요소를 복합 서비스의 제어 흐름으로 매핑한다. 데이터 관점 컴포넌트의 경우, 엔티티 클래스를 제어하는 클래스를 포함한 클래스 집합을 서비스로 매핑한다. 여기서 서비스 인터페이스를 이용하여 기능을 공개하는 후보 오퍼레이션 집합은 엔티티 클래스를 제어하는 클래스의 오퍼레이션 집합에서 도출한다.

전체매핑 03. Provided 인터페이스에서 서비스 인터페이스로의 변환: 비즈니스 로직과 데이터 관점 컴포넌트의 인터페이스 모두 서비스 인터페이스로 매핑한다. CBD에서 비즈니스 로직 컴포넌트의 워크플로우는 마크로(macro) 워크



(그림 5) 컴포넌트와 서비스의 매핑 관계



(그림 6) Provided 인터페이스에서 서비스 인터페이스로의 변환

플로우와 마이크로(micro) 워크플로우로 나뉜다. 마크로 워크플로우는 상위 수준의 컴포넌트 간 흐름이고, 마이크로 워크플로우는 클래스나 내부 컴포넌트 간 흐름으로 비교적 상세하게 기술된다. 4장 2절에서 언급하였듯이, SOC에서는 마크로 워크플로우는 비즈니스 프로세스에 매핑되고 제어된다. 그러므로 본 절에서는 비즈니스 로직 컴포넌트의 마이크로 워크플로우와 데이터 관점 컴포넌트의 인터페이스만 고려하면 된다. (그림 6)에서 보는 바와 같이, 비즈니스 로직 컴포넌트의 마이크로 워크플로우는 로직(logic)이나 연산(computation)을 처리하는 서비스의 인터페이스와 매핑된다. 그리고 데이터 관점 컴포넌트의 인터페이스는 데이터를 관리하는 서비스 컴포넌트의 인터페이스와 매핑된다.

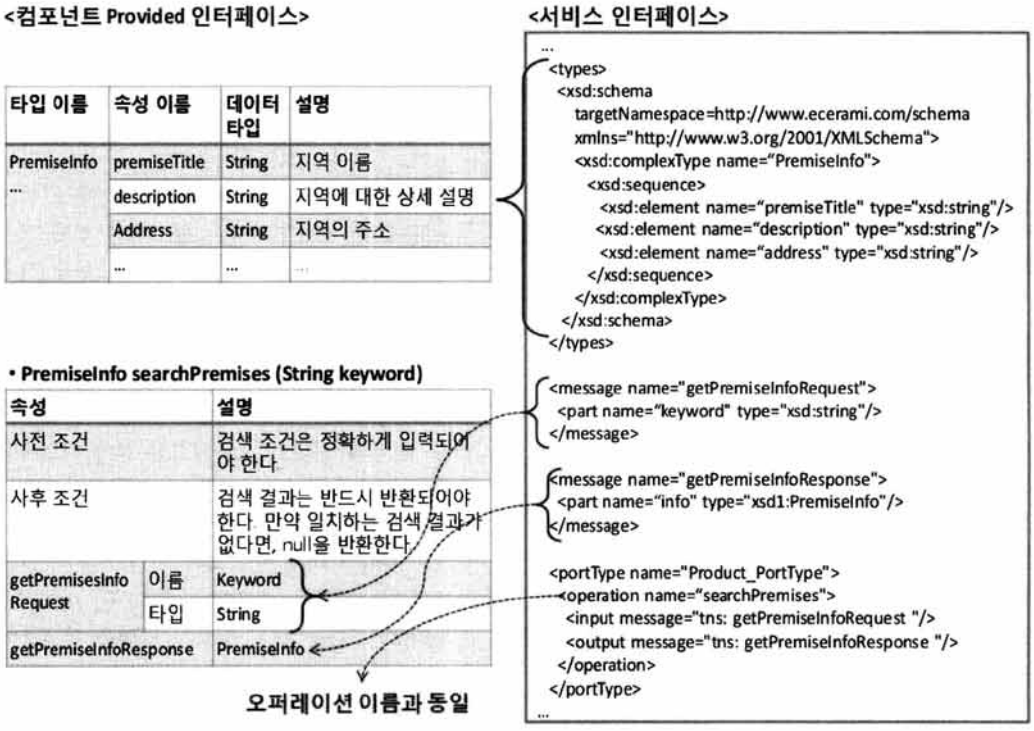
이를 위해 다음과 같은 세 가지 스텝을 적용할 수 있다. 첫째, 각 비즈니스 로직 컴포넌트의 마이크로 워크플로우를 식별하고, 로직을 처리하는 서비스 컴포넌트의 인터페이스로 매핑한다. 둘째, 각 데이터 관점 컴포넌트의 인터페이스를 식별하고 데이터를 관리하는 서비스 컴포넌트의 인터페이스로 매핑한다. SOC에서 서비스 인터페이스는 WSDL로 명세되고, UDDDI에 등록된다. 먼저, 비즈니스 프로세스로부터 인터페이스가 정의되는데, 이 인터페이스는 서비스 컴포넌트가 반드시 준수해야 하는 것을 정의하고 있다. 컴포넌트 인터페이스와 서비스 컴포넌트의 인터페이스로 예상되는 것과 사이에 차이가 생길 수 있다. 그러므로 마지막 스텝에서는 컴포넌트 인터페이스의 시그니처 조절하여 서비스 컴포넌트 인터페이스의 시그니처에 알맞게 해야 한다. 이는 이름을 다시 명명하거나, 매개변수를 묶거나, 데이터 타입을 조절하거나, 매개변수 리스트를 제배치하는 것으로 가능하다.

(그림 7)은 Provided 인터페이스에서 서비스 인터페이스로의 변환하기 위한 유도 과정을 보여준다. 유도 시 WSDL에서 매개변수를 명세하는 표준 방법을 준수해야 한다. 인터페이스 오퍼레이션(operation)은 <interface> 태그(tag) 안에서 <operation name = "">으로 표현할 수 있다. 매개변수 이름은 <operation> 태그 안에서 <input label="in" element="">로, 반환 메시지의 이름은 <output label="out" element="">로 표현 가능하다. 그리고 오퍼레이션, 매개변수, 반환 메시지의 타입은 <type>이나 <complexType>을 이용하여 기술할 수 있다.

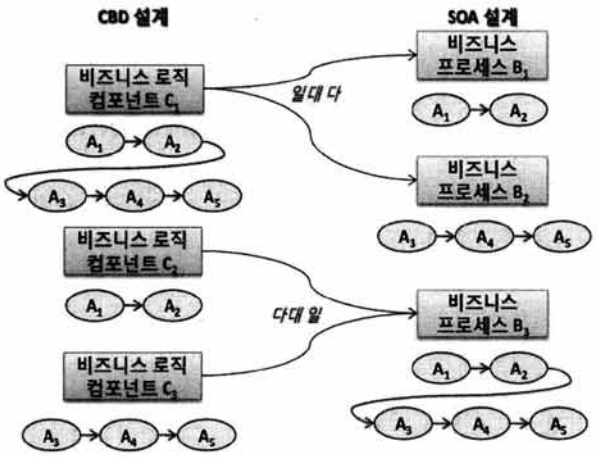
5.2 부분 매핑을 위한 변환 기법

부분매핑 01. 비즈니스 로직 컴포넌트에서 비즈니스 프로세스로의 변환: 서비스는 작은 단위의 액티비티 간의 워크플로우로 명세되는 비즈니스 프로세스와 함께 정의된다. 또한, 워크플로우는 BPEL과 같은 비즈니스 프로세스 모델링(modeling) 언어로 명세된다[14]. CBD 측면에서는, 비즈니스 로직 컴포넌트가 객체나 다른 컴포넌트 간의 메소드 호출 순서를 조정하는 기능을 한다. 그러므로 우리는 비즈니스 로직 컴포넌트 내부에 설계되어 있는 워크플로우를 분석하여 비즈니스 프로세스를 유도할 필요가 있다. 이상적으로는, 비즈니스 로직 컴포넌트의 워크플로우가 비즈니스 프로세스와 일대 일 매핑 관계일 수 있다. 실무에서는 (그림 8)과 같이 일대 다, 다대 일의 매핑 관계도 가능하다.

첫 번째 케이스는 하나의 비즈니스 로직 컴포넌트의 워크플로우가 여러 개의 비즈니스 프로세스의 워크플로우로 매핑되는 경우이다. 그러므로 소스 워크플로우는 여러 개의



(그림 7) Provided 인터페이스에서 서비스 인터페이스로의 변환 예



(그림 8) 비즈니스 로직 컴포넌트에서 비즈니스 프로세스로의 변환 케이스

워크플로우로 분할되어야 한다. 두 번째 케이스는 반대 경우로 여러 컴포넌트의 워크플로우가 하나의 비즈니스 프로세스의 워크플로우로 결합된다.

이를 위해 다음과 같은 세 가지 스텝이 있다. 첫째, 각 서비스로부터 비즈니스 로직 컴포넌트를 식별한다. 둘째, 비즈니스 로직 컴포넌트 내부에 설계되어 있는 워크플로우를 추출한다. 비즈니스 로직 컴포넌트를 위해 모델링 한 시퀀스 다이어그램(sequence diagram)과 같은 커뮤니케이션 다이어그램이 워크플로우를 추출하는 데에 직접적인 소스가 될 수 있다. 마지막으로는, 이런 정보를 이용하여 서비스 단계의 워크플로우를 작성하고 명세한다. 워크플로우를 작성할 때에, 다음과 같은 여러 가지 요소를 고려해야만 한다.

- 단일 트랜잭션(transaction) vs. 오래 지속되는 트랜잭션: 단일 트랜잭션의 경우는 관련 선후 액티비티의 흐름을 분리하여 비즈니스 프로세스로 매핑할 수 있으나 오래 지속되는 트랜잭션의 경우 관련 선후 액티비티의 흐름을 분리할 수 없다.
- 롤백(rollback): 롤백이 수행되었을 때 실행되는 모든 액티비티는 하나의 제어 흐름으로 정의되어야 한다.
- 메소드 호출의 동기화 vs. 메소드 호출의 비동기화: 메소드 호출이 동기화될 경우, 동기화와 관련된 액티비티는 하나의 제어 흐름으로 정의되어야 한다. 반면 메소드 호출이 비동기화일 경우, 목적에 따라 제어흐름을 나눌 수 있다.

비즈니스 로직 컴포넌트의 워크플로우는 비즈니스 프로세스에서 활용할 수 있으나, 추가적인 설계가 필요하다. 즉, 비즈니스 프로세스에서 사용하는 서비스와 비즈니스 프로세스 관계를 정의하기 위한 PartnerLink, 비즈니스 프로세스에서 사용하는 프로세스 변수, 서비스 간의 연관 관계에 따라 발생하는 변수 매핑을 위한 Assign 액티비티 등을 추가하여야 한다[15]. 또한, 비즈니스 로직 컴포넌트의 워크플로우는 재사용을 기반으로 설계되었으나, 비즈니스 프로세스의 제어 흐름은 특정 문맥(context)에 맞게 설계되어 있다. 그러므로 특정문맥에 맞게 재사용 가능한 워크플로우를 특화시키고, 문맥 특화적인 설계를 추가하여야 한다. 즉, 특정 문맥에 맞게 비즈니스 프로세스에 추가적인 액티비티를 추가하여야 한다.

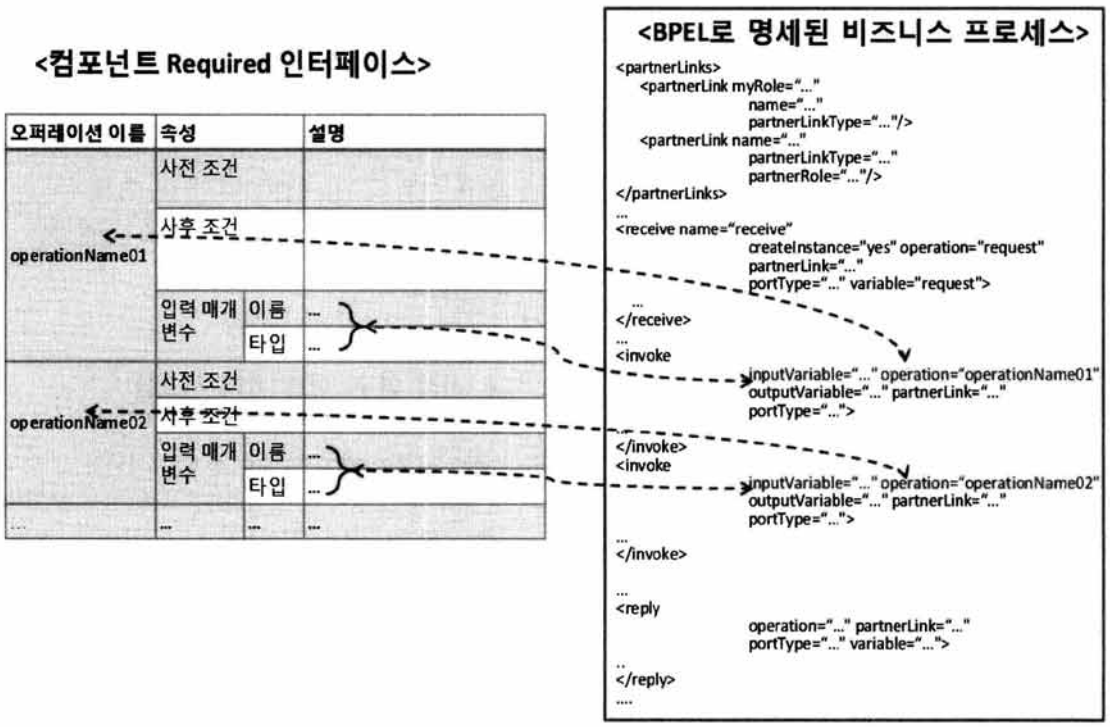
부분매핑 02. 비즈니스 로직 컴포넌트에서 복합 서비스의 워크플로우로의 변환: 비즈니스 로직 컴포넌트에서 복합 서비스의 워크플로우로의 변환은 부분매핑 01의 경우와 비슷하다. 단, 비즈니스 로직 컴포넌트와 마찬가지로 복합 서비스는 재사용을 목적으로 설계 및 구현되기 때문에 특정 문맥에 맞게 특화시키는 작업은 수행하지 않아도 된다.

부분매핑 03. 데이터 관점 컴포넌트에서 서비스 컴포넌트로의 변환: 데이터 관점 컴포넌트는 데이터베이스의 데이터를 관리하기 위한 질의를 담당하는 클래스와 이런 질의를 제어하는 클래스로 나뉜다. 질의를 담당하는 클래스의 경우 데이터베이스에 접속하고 질의를 전송하기 위한 JDBC 드라이버와 질의 자체를 담당하는 오퍼레이션을 포함한다. 질의를 제어하는 클래스는 컴포넌트의 Provided 인터페이스와 매핑될 수 있도록 질의의 추상화 레벨을 향상시켜서 오퍼레이션을 정의한다.

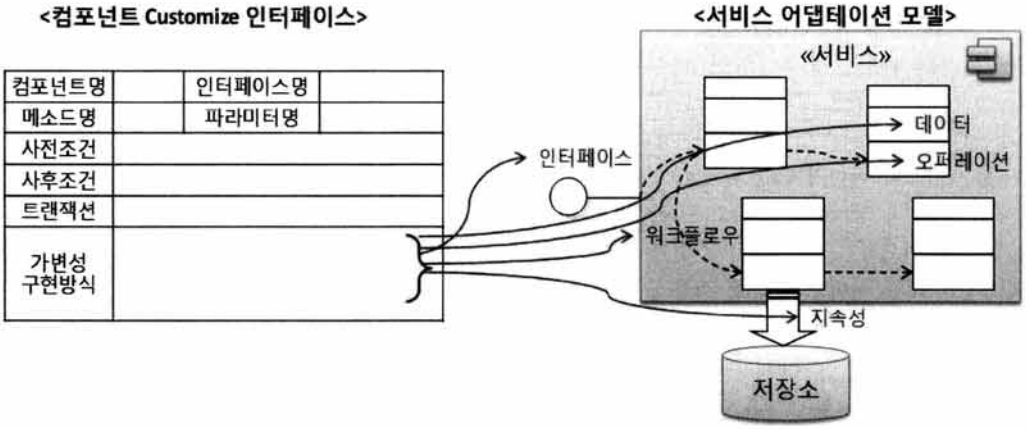
서비스 역시 대부분 데이터를 다루고 있기 때문에 서비스의 데이터를 저장하고 있는 데이터베이스의 데이터를 관리하기 위한 질의를 담당하는 클래스와 이런 질의를 제어하는 클래스를 포함한다. 그러나 서비스의 경우 질의만을 위한 기능성을 제공하지 않고, 이런 질의를 통해 얻을 수 있는 사용자 관점의 기능성이 필요하다. 또한, 데이터 관점 컴포넌트에서 제공하고 있는 질의를 담당하는 클래스에서 사용하고 있는 JDBC 드라이버의 경우 서비스 컴포넌트에 부합될 수도 있고 아닐 수도 있다. 그러므로 데이터 관점 컴포넌트에서 정의하고 있는 질의를 위한 오퍼레이션만 서비스 컴포넌트로 변환이 가능하다.

부분매핑 04. Required 인터페이스에서 비즈니스 프로세스로의 변환: 컴포넌트 Required 인터페이스는 특정 기능의 수행을 위해서 다른 컴포넌트의 기능성을 사용하기 위하여 정의한다. 즉, Required 인터페이스는 다른 컴포넌트의 호출을 암시적으로 명세한다.

SOA 설계에서 비즈니스 프로세스는 여러 서비스 간의 호출을 제어하기 위한 흐름을 포함한다. 이 흐름에는 관련 있는 두 개의 서비스 오퍼레이션 간의 입력 및 출력의 연관 관계를 포함한다. 이런 연관 관계를 정의하기 위하여 BPEL로 명세된 비즈니스 프로세스의 경우, 중재자 역할을 한다. 즉, 두 개의 서비스 오퍼레이션 간의 호출 관계는 비즈니스 프로세스를 거쳐서 수행되게 된다. 즉, 비즈니스 프로세스에는 서비스 호출을 위해 필요한 입력을 명세하여야 한다. (그림 9)는 Required 인터페이스에 명세되어 있는 정보를 활용하여 비즈니스 프로세스에 필요한 정보를 유도하는 것을 보여준다. 즉, Required 인터페이스에 있는 오퍼레이션 이름은



(그림 9) Required 인터페이스에서 비즈니스 프로세스로의 변환



(그림 10) Customize 인터페이스에서 서비스 어댑테이션 모델로의 변환

<invoke>라는 엘리먼트 안에 기술하여야 하는 operation 어트리뷰트에 매핑되고, 입력 매개 변수는 inputVariable 어트리뷰트에 매핑된다. 그러나 Required 인터페이스 정보의 활용은 비즈니스 프로세스가 이미 도출되어 있다는 가정에서 활용가능하기 때문에 비즈니스 프로세스의 워크플로우 자체를 정의하는 데에는 도움이 되지 않는다. 즉, 비즈니스 프로세스의 워크플로우가 결정되어 있는 상태에서 상세 정보를 기술하는 데에 이 변환이 활용된다.

부분매핑 05. Required 인터페이스에서 복합 서비스 워크플로우로의 변환: Required 인터페이스에서 복합 서비스 워크플로우로의 변환은 부분매핑 04의 경우와 비슷하다. 비즈니스 프로세스와 복합 서비스의 워크플로우의 설계 결과물은 동일하게 BPEL 문서와 WSDL 문서이기 때문이다. 즉, 부분매핑 04에서 언급한 바와 같이 Required 인터페이스 명세서에서 인터페이스를 구성하고 있는 오퍼레이션 이름, 입력 매개 변수의 이름과 타입이 워크플로우로 매핑될 수 있다. 그러므로 변환을 하기 위한 수행 절차 및 내용은 부분매핑 04와 동일하다. 단, 부분매핑 04와 다른 점은 매핑된 결과물인 복합 서비스 워크플로우가 재사용 가능하다는 점이다.

부분매핑 06. Customize 인터페이스에서 서비스 어댑테이션 모델로의 변환: 컴포넌트 인터페이스 중 커스터마이즈 인터페이스의 경우, 현재 서비스 인터페이스를 명세하는 대표적인 표준인 WSDL에서 명세할 수 있는 방법을 제공하고

있지 않다. 반면, Customize 인터페이스에서 컴포넌트를 특화하기 위하여 제공하는 오퍼레이션 집합들은 컴포넌트의 가변성 지원 정도를 요약해서 보여준다. 즉, (그림 10)에서처럼 Customize 인터페이스를 포함하고 있는 컴포넌트, 인터페이스 이름, 인터페이스 내의 메소드 이름 등을 명세하고 있다.

이 정보 중 가변성 구현방식에 기입되어 있는 정보가 서비스 어댑테이션 모델의 각 어댑테이션 할 수 있는 지점으로 매핑된다. 즉, 가변성 구현 방식에는 가변점 유형, 가변치 유형, 가능한 가변치 등이 명세된다. 관련 내용은 서비스 어댑테이션에서 어댑테이션을 수행하여야 할 지점, 어댑테이션 가능한 범위 등을 명세하는 데 사용될 수 있다.

5.3 매핑되지 않는 요소를 위한 설계 기법

단일 서비스 및 복합 서비스를 구현을 위해 설계 정보는 모두 매핑이 가능하지만, SOA 설계 및 구현을 위해서는 SOA 표준에 따라서 인터넷을 통하여 서비스를 사용할 수 있도록 서비스화 하는 요소들이 존재한다.

단일 서비스를 위한 서비스 인터페이스 정의: 서비스를 명세함에 있어서 일반적으로 WSDL 표준을 따른다. WSDL은 4개의 주요 엘리먼트인 <types>, <message>, <portType>, <bindings>로 구성된다. 이 엘리먼트 중에 <types>, <message>, <portType>는 컴포넌트의 Provided 인터페이스에 기술된 내용에서 매핑된다. <bindings>는 단일 서비스를 호출하기 위해 사용하여야 할 프로토콜, 호출할 수 있는 엔드포인트(endpoint) 주소 등을 명세하여야 한다.

복합 서비스를 위한 비즈니스 프로세스 및 서비스 인터페이스 정의: 단일 서비스와 마찬가지로, WSDL 인터페이스는 서비스 소비자가 BPEL로 정의된 비즈니스 프로세스를 접근하고 호출하기 위해 필요하다. 다른 서비스들과 상호 작용을 한다. BPEL에서 복합 서비스와 상호 작용하는 모든 파트너는 <partnerLinks> 엘리먼트에 정의된다. <receive>와 <reply>액티비티에서 명세된 partnerLink 어트리뷰트는 서비스 소비자가 비즈니스 프로세스를 호출하고 프로세스로부터 응답을 얻는 것을 가능하게 해준다. 그리고 <invoke> 액티비티에서 명세된 partnerLink속성은 비즈니스 프로세스가 다른 서비스와 상호 작용하는 것을 가능하게 해준다. 이와 동시에 복합서비스를 위한 WSDL에서는 <partnerLinkType>을 이용하여 복합 서비스에 참여하는 모든 파트너들에 대한 정보를 <partnerLinks> 에 정의된 정보와 동일하게 선언해야 한다. 또한, 복합 서비스를 호출하기 위해 사용하여야 할 프로토콜, 호출할 수 있는 엔드포인트 주소 등을 명세하여야 한다.

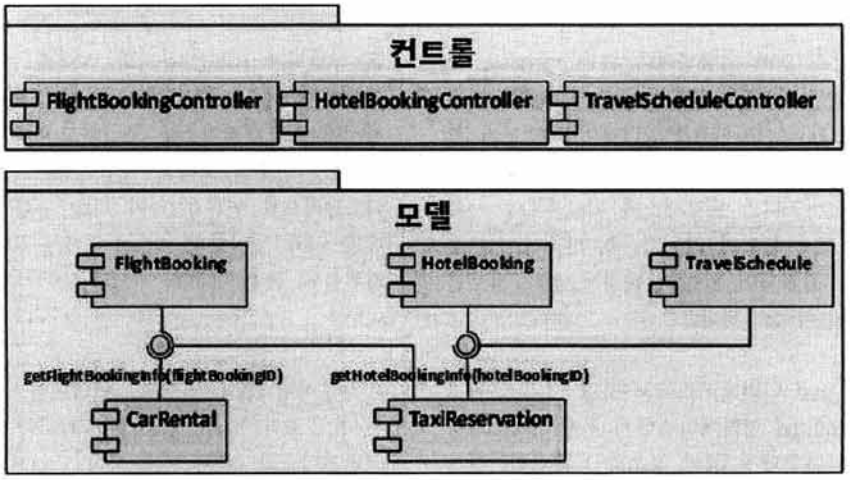
6. 사례 연구: 여행 관리 시스템

본 장에서는 사례 연구를 통하여 제안한 변환 기법의 적용 범위 및 효율성을 보여준다. 사례 연구의 도메인으로는 여행 관리 시스템을 선정하였다. 현재 서비스되고 있는 여러 여행 관리 시스템[16][17][18]을 살펴본 결과, 모듈화되고 재사용 가능한 단위의 여러 기능성을 제공하고 있고, 이런 기능성을 요구에 따라 특화시켜야 하는 부분이 있으며, 일련의 기능성들을 수행하기 위한 워크플로우를 제공하고 있다. 즉, 컴포넌트 기반으로 개발하거나, 서비스 기반으로 개발하기에 적합한 특징을 갖고 있기 때문에 사례 연구의 도메인으로 선정하였다.

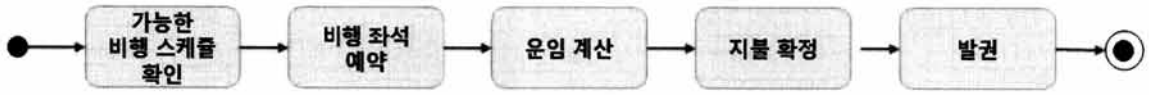
여행 관리 시스템의 주요 기능은 비행 예약, 호텔 예약, 차량 예약 혹은 택시 예약, 일정 관리로 이루어진다. 그러므로 이런 기능을 수행하기 위하여 (그림 11)에서처럼 컨트롤 계층의 3가지 컴포넌트와 모델 계층의 5가지 컴포넌트가 필요하다. 여기서의 계층 구조는 CBD 설계에서 일반적으로 사용하는 MVC 아키텍처 스타일을 활용하였다.

컨트롤 계층의 3개의 비즈니스 로직 컴포넌트는 각 기능성에 맞게 필요한 데이터를 가져오고 처리할 수 있도록 워크플로우를 포함하고 있다. 예를 들어, FlightBooking Controller는 비행 예약을 위하여 (그림 12)처럼 5개 단계로 구성된 비행기 예약하는 워크플로우를 포함하고 있다.

모델 계층의 5개의 데이터 관점 컴포넌트는 주로 컨트롤



(그림 11) 여행 관리 시스템을 위한 컴포넌트 다이어그램



(그림 12) FlightBookingController 컴포넌트의 액티비티 다이어그램

계층의 컴포넌트에서 요청하는 데이터를 관리하는 역할을 한다. 예를 들어 FlightBooking 컴포넌트는 소비자가 비행기를 예약하기 위해 질의하거나 정보를 등록하기 위하여 비행 스케줄 검색, 비행기 좌석 검색, 비행 예약 등의 기능을 제공한다. 또한, 소비자가 필요한 정보를 등록하기 위하여 비행기 정보 등록, 좌석 정보 등록, 비행 스케줄 등록과 같은 기능을 제공한다. 이런 기능을 위하여 FlightBooking 컴포넌트는 6개의 엔티티 클래스(Plane, FlightSeat, FlightEvent, Booking, Customer, Payment)로 구성된다. <표 1>은 FlightBooking 컴포넌트의 인터페이스를 보여준다. 이 컴포넌트는 8개의 제공 인터페이스와 2개의 커스터마이즈 인터페이스로 구성된다.

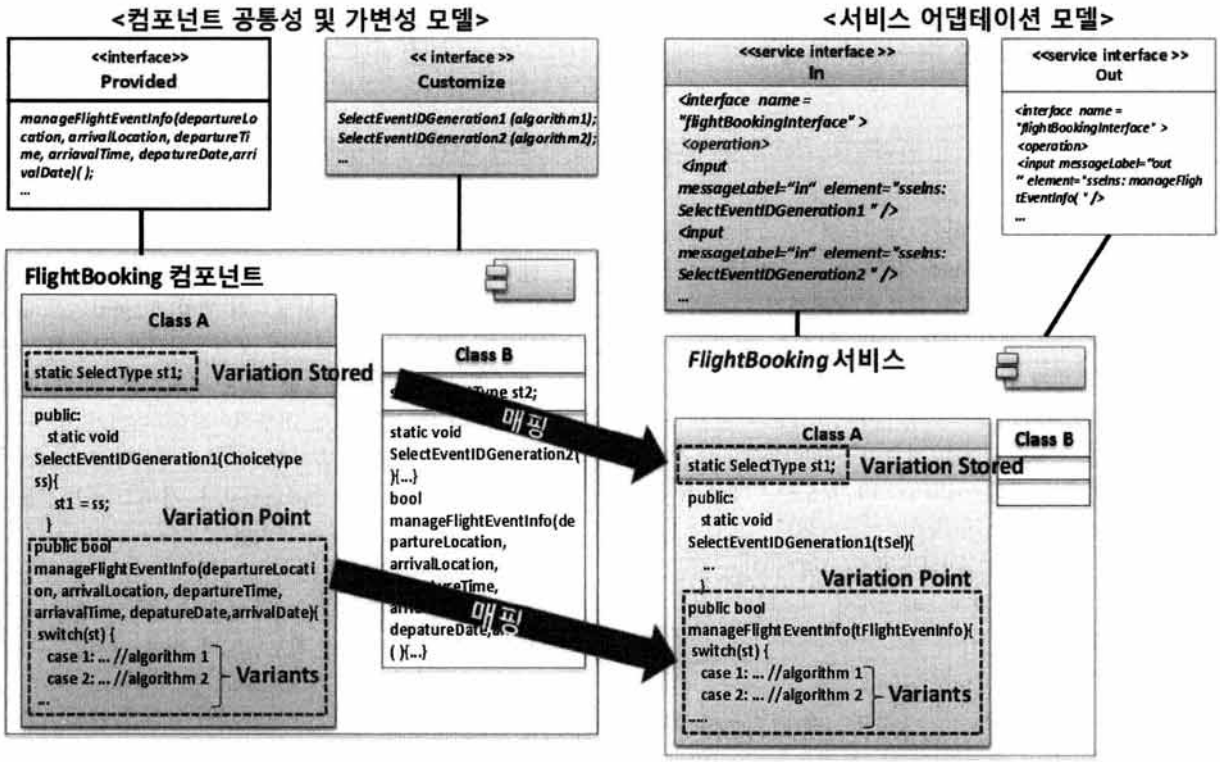
6.1 전체매핑을 위한 변환 기법 적용

전체매핑 01. 여행 관리 시스템 공통성 및 가변성 모델에서 서비스 어댑테이션 모델로의 변환: FlightBooking 컴포넌트는 두 개의 커스터마이즈 오퍼레이션을 포함하고 있다. 이 오퍼레이션은 컴포넌트를 위한 가변치를 할당하기 위한 것이다. SOC에서 서비스 어댑테이션이 어떻게 발생하는지 의견의 일치가 아직 미흡하다. 서비스 컴포넌트에 해당하는 Provided 오퍼레이션을 추가하였으나, 서비스 가변성을 커스터마이즈하기 위한 목적으로 사용한다.

(그림 13)은 전체매핑 01에 해당하는 컴포넌트 공통성 및 가변성 모델에서 서비스 어댑테이션 모델로의 변환 결과 중 하나이다. FlightBooking 컴포넌트에 비행 정보를 관리하는

〈표 1〉 FlightBooking 컴포넌트 인터페이스 명세

오퍼레이션 이름(시그니처)	설명	유형
managePlane (company, name)	이 인터페이스는 회사, 비행기 식별 ID 등과 같은 비행기 정보를 관리하는데 사용한다. 여기서 비행기 식별 ID의 경우 자동적으로 생성된다. 사전조건: 현재 존재하는 비행기 정보를 사용하여야 한다.	Provided
manageFlightSeatInfo (seatPrice, seatGrade, seatNo)	이 인터페이스는 좌석 가격, 좌석 등급 (1등석, 비즈니스석, economy 석)등과 같은 좌석 정보를 관리하는데 사용한다. 사전조건: 현재 존재하는 좌석 정보를 사용하여야 한다.	Provided
manageFlightEventInfo (departureLocation, arrivalLocation, departureTime, arriavalTime, depatureDate,arrivalDate)	이 인터페이스는 출발 시간, 도착 시간, 출발 국가, 도착국가와 같은 비행 정보를 관리하는데 사용한다. 사전조건: 비행기와 좌석 정보가 먼저 등록되어 있어야 한다.	Provided
bookFlight (planeID, flightSeatID, flightEventID)	이 인터페이스는 비행기, 좌석, 비행 정보를 기반으로 예약을 수행하는데 사용한다. 또한, 이 인터페이스는 탑승 기록을 남기는 데에도 사용한다 사전조건: 고객 정보가 유효한 정보이어야 한다. 사후조건: 날짜, 시간, 출발 및 도착 장소가 올바른 정보이어야 한다.	Provided
getPlaneInfo(planeID)	이 인터페이스는 비행기 정보를 알려주기 위하여 사용한다	Provided
getFlightSeatInfo(seatID)	이 인터페이스는 좌석 정보를 알려주기 위하여 사용한다.	Provided
getFlightEventInfo (flightEventID)	이 인터페이스는 비행 정보를 알려주기 위하여 사용한다.	Provided
manageCustomer(travelerID)	이 인터페이스는 탑승자 이름, 주소 등과 같은 정보를 관리하기 위하여 사용한다.	Provided
selectPlaneIDType (selectedType)	이 인터페이스는 비행기 ID 유형을 문자열, 숫자 등으로 변환하기 위하여 사용한다.	Customize
selectEventIDGeneration (algorithmID)	이 인터페이스는 비행 이벤트 ID를 만들기 위하여 사용하는 방법을 특화시키기 위하여 사용한다.	Customize



(그림 13) 전체매핑 01에 따른 변환 결과

오퍼레이션이 있는데 이 부분에서 각 항공사마다 다르게 관리할 수 있기 때문에 관련 가변성이 발생한다. 이렇게 설계된 가변성은 FlightBooking 서비스의 오퍼레이션에도 똑같이 설계 및 구현된다. 즉, 인터페이스 명세 양식은 다르지만 가변성을 위한 내부 클래스 설계 및 구현은 매핑된다.

그림에서 다루고 있지는 않지만, FlightBooking 컴포넌트를 설계하는데 사용한 공통성 및 가변성 모델은 FlightBooking 서비스뿐 아니라 Traveler 서비스 및 Payment 서비스 어댑테이션 모델에도 매핑되었다. 예를 들어 FlightBooking 컴포넌트에서 항공료를 계산하는 로직 부분은 회사마다 다르지만, 관련 기능은 공통적으로 사용된다. 즉, 항공료를 계산하는 로직 부분이 가변점으로 설계되었고, 실제 로직은 가변치로 동적으로 할당할 수 있도록 플러그인 기법으로 설계 및 구현되었다. 이런 설계 및 구현은 Payment 서비스에서도 동일하게 활용될 수 있었다. 즉, Payment 서비스에서는 항공사마다 원하는 다른 항공료 계산 로직을 추가하여 어댑테이션을 수행할 수 있도록 어댑테이션 모델에 설계되었다.

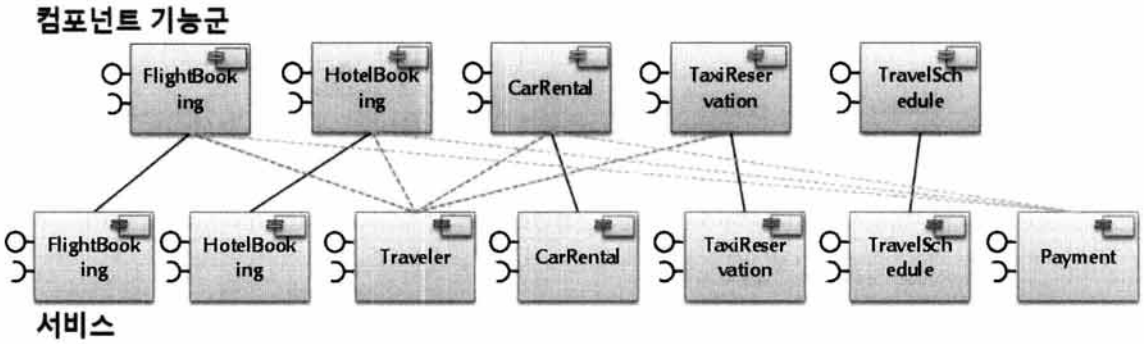
전체매핑 02. 여행 관리 시스템 컴포넌트에서 서비스로의 변환: 먼저 여행 관리 시스템 컴포넌트에서 서비스로의 변환을 하기 위하여, 비즈니스 로직 컴포넌트와 데이터 관점 컴포넌트로 나뉘어져 있는 컴포넌트들의 관계를 정의한다. <표 2>에서처럼 하나 이상의 관련된 컴포넌트를 모아서 컴포넌트 기능군으로 모은다. 그런 다음 이름, 기능, 제약사항을 이용하여 상위의 컴포넌트 설명을 작성한다.

〈표 2〉 컴포넌트 기능군 요약표

컴포넌트	컴포넌트 기능군	기능성	제약사항
FlightBooking Controller	Flight Booking	비행 예약을 위한 기능을 제공함	트랜잭션 처리 요구사항: 10,000 / 1 min
FlightBooking			
...	...	...	...

그런 다음, 컴포넌트 및 서비스 별로 매핑 관계를 표시하기 위하여 다중성을 추가한다. 예를 들어서 FlightBooking 컴포넌트 기능군과 같은 경우는 FlightBooking, Traveler, Payment 서비스로 매핑이 되었습니다. 이런 경우 FlightBooking 컴포넌트 기능군은 세 개의 서비스와 연관이 있으므로 이에 따른 다중성을 정의하자면 하나의 컴포넌트 기능군이 세 개의 서비스와 연관이 있다. 즉, 컴포넌트 기능군과 서비스의 관계는 일대 다의 관계로 정의된다.

또한, 5개의 컴포넌트 기능군에서 사용하고 있는 기능성이 추출되어 하나의 서비스 Payment가 추가적으로 도출되었고, 또한 5개의 컴포넌트 기능군을 제어할 수 있는 기능성이 추가적으로 필요하였기에 다른 하나의 서비스 Traveler가 추가적으로 도출되었습니다. 여기서 언급된 2개의 서비스 경우 여러 개의 컴포넌트의 일부 기능들이 추출되어 서비스로 구성되었기 때문에 컴포넌트 기능군과 서비스의 관계는 다대 일로 정의되었습니다. 여기서 언급된 2개



(그림 14) 컴포넌트에서 서비스로의 변환 결과

의 서비스 경우 여러 개의 컴포넌트의 일부 기능들이 추출되어 서비스로 구성되었기 때문에 컴포넌트 기능군과 서비스의 관계는 다대 일로 정의된다. 그러므로 여행 관리 시스템에서 컴포넌트 기능군과 서비스는 (그림 14)와 같이 다대다 관계로 정의되었다.

추가된 서비스에서의 변환은 다음과 같다. 먼저, Traveler 서비스의 경우는 복합 서비스로써 FlightBooking, Hotel Booking, CarRental, TaxiReservtion 서비스에서 제공하는 기능성을 재사용하여 서로 간의 제어 흐름을 포함한다. 그러므로 각 컴포넌트에서 필요한 정보를 가져와서, 이를 기반으로 정의하여야 하기 때문에 여러 컴포넌트와 매핑된다고 할 수 있다. 두 번째, Payment 서비스의 경우는 여러 컴포넌트에서 모두 포함하고 있는 기능성을 재사용 가능하게 도출한 경우이다. 즉, 5개의 컴포넌트 기능군에서는 지불 기능을 각 컴포넌트의 기능에 맞게 포함하고 있다. 여기에서의 지불 기능은 각 컴포넌트의 기능에 맞게 요금 계산부터 과금까지 포함한다. 그러나 서비스로 매핑할 때는 요금 계산 이후의 과정만을 포함하여 재사용 가능할 수 있도록 매핑된다.

전체매핑 03. 여행 관리 시스템 Provided 컴포넌트 인터페이스에서 서비스 인터페이스로의 변환: <표 1>에서처럼 FlightBooking 컴포넌트의 인터페이스는 8개의 제공 인터페이스와 2개의 커스터마이즈 인터페이스로 구성되어 있다. 예를 들어, 제공 인터페이스 중 bookFlight(planeID, flightSeatID, flightEventID)를 변환한다고 하자. bookFlight는 서비스 인터페이스의 오퍼레이션에 매핑된다. 이 오퍼레

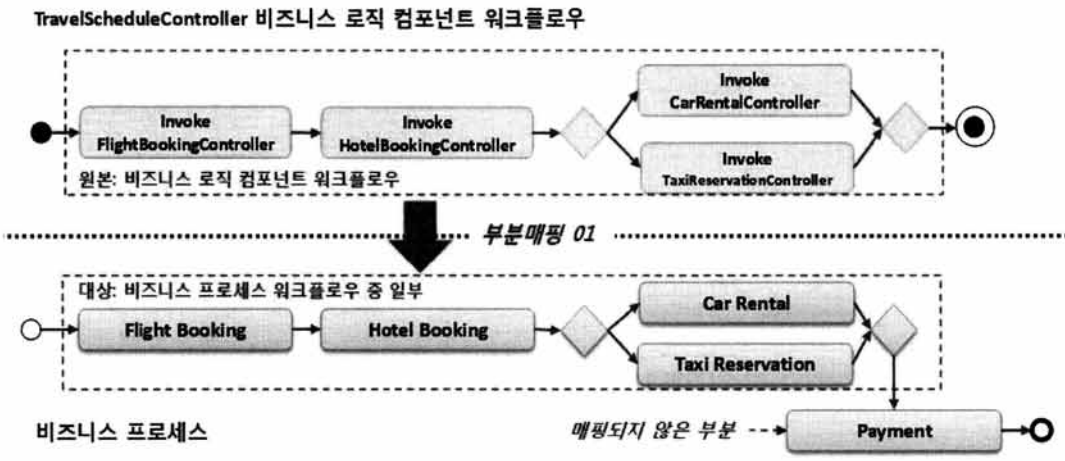
<컴포넌트 Provided 인터페이스>

오퍼레이션 이름(시그니처)	설명	유형
bookFlight (planeID, flightSeatID, flightEventID)	이 인터페이스는 비행기, 좌석, 비행 정보를 기반으로 예약을 수행하는데 사용한다. 또한, 이 인터페이스는 탑승 가족을 남기는 데에도 사용한다 사전조건: 고객 정보가 유효한 정보이어야 한다. 사후조건: 날짜, 시간, 출발 및 도착 장소가 올바른 정보이어야 한다.	Provided

<서비스 인터페이스>

```
<types>
  <xs:schema
    ....
   <xs:element name="bookFlight" type="tbookFlight"/>
   <xs:complexType name="tbookFlight">
    <xs:sequence>
      <xs:element name="planeID" type="xs:string"/>
      <xs:element name="seatID" type="xs: string "/>
      <xs:element name="eventID" type="xs:string "/>
</xs:sequence>
   </xs:complexType>
...
  </xs:schema>
</types>
<interface name = "flightBookingInterface" >
   <operation name="bookFlight" pattern="http://www.w3.org/2004/03/wsdl/
out" >
     <input messageLabel="out"
        element="sseIns:getPlaneInfo" />
     ...
</operation>
</interface>
```

(그림 15) FlightBooking 컴포넌트 인터페이스에서 서비스 인터페이스로의 변환



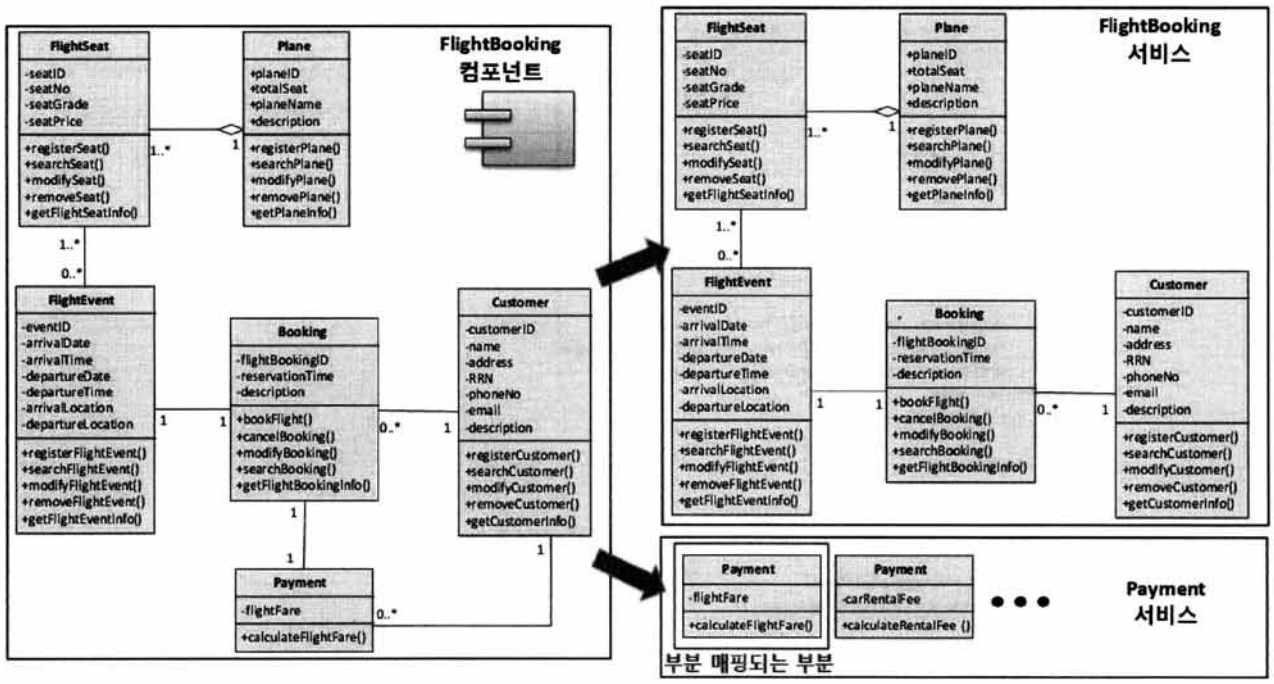
(그림 16) TravelerScheduleController를 기반으로 한 비즈니스 프로세스 유도

이션에 필요한 메시지는 tbookFlight로 정의하고, 관련 타입은 컴포넌트 제공 인터페이스의 시그니처인 planeID, flightSeatID, flightEventID와 매핑할 수 있도록 정의한다. 또한, 이 오퍼레이션의 경우는 반환 값이 있도록 정의되어 있으므로, 오퍼레이션의 패턴을 in-out으로 명세한다. 이에 따른 변환 결과는 (그림 15)와 같다.

6.2 부분 매핑을 위한 변환 기법 적용

부분매핑 01. 여행 관리 시스템 비즈니스 로직 컴포넌트에서 비즈니스 프로세스로의 변환: 먼저 여행 관리 시스템의 CBD 모델에 정의된 세 개의 비즈니스 로직 컴포넌트와 두 개의 컨트롤 클래스를 기반으로 비즈니스 프로세스 변환 과정을 수행한다. 이를 위해 UML 액티비티(activity) 다이어그램을 이용하여 비즈니스 로직 컴포넌트와 컨트롤 클래스 내부에 설계되어 있는 워크플로우를 추출한다.

(그림 16)은 TravelerScheduleController를 기반으로 한 비즈니스 프로세스 유도를 보여준다. 먼저, TravelSchedule Controller는 4개의 다른 비즈니스 로직 컴포넌트를 실행하는 제어 흐름을 포함하고 있다. 이를 기반으로 하여 비즈니스 프로세스의 워크플로우를 정의한다. 워크플로우를 정의하기에 앞서 이미 우리는 각 컴포넌트 기능군을 서비스로 매핑하였기 때문에 관련 기능을 호출하는 것을 일대 일로 매핑할 수 있었다. 단, Payment 서비스의 경우는 모든 컴포



(그림 17) FlightBooking 데이터 관점 컴포넌트에서 FlightBooking 및 Payment 서비스로의 변환

넌트에서 재사용할 수 있는 것이어서 추가적인 서비스로 추출하였으며, 이에 따라, 관련 비즈니스 프로세스에서도 Payment 서비스 호출하는 것을 추가하였다.

부분매핑 03. 여행 관리 시스템 데이터 관점 컴포넌트에서 서비스 컴포넌트로의 변환: 전체매핑 02를 통해서 수행한 것을 기반으로 데이터 관점 컴포넌트를 서비스 컴포넌트로 변환한다. 예를 들어, (그림 17)은 FlightBooking 데이터 관점 컴포넌트의 변환 결과를 보여준다.

FlightBooking과 관련 있는 데이터의 관리와 관련있는 클래스만 먼저 매핑한다. 클래스 매핑시 오퍼레이션이 질의와 관련있는 것인지 확인하여야 한다. 그런 다음, 전체매핑 02의 결과에 따라 Payment 관련된 클래스는 FlightBooking에서 제외하였다. 그리고 제외된 Payment 클래스는 Payment 서비스를 구성하는 클래스 중 하나로 매핑되었다. 이 변환 과정을 통하여서 FlightBooking 데이터 관점 컴포넌트가 2개의 서비스와 매핑됨을 확인할 수 있었다.

부분매핑 06. 여행 관리 시스템 Customize 인터페이스와 서비스 어댑테이션 모델: (그림 18)은 FlightBooking 컴포넌트 Customize 인터페이스에서 서비스 어댑테이션 모델로의 변환의 예를 보여준다.

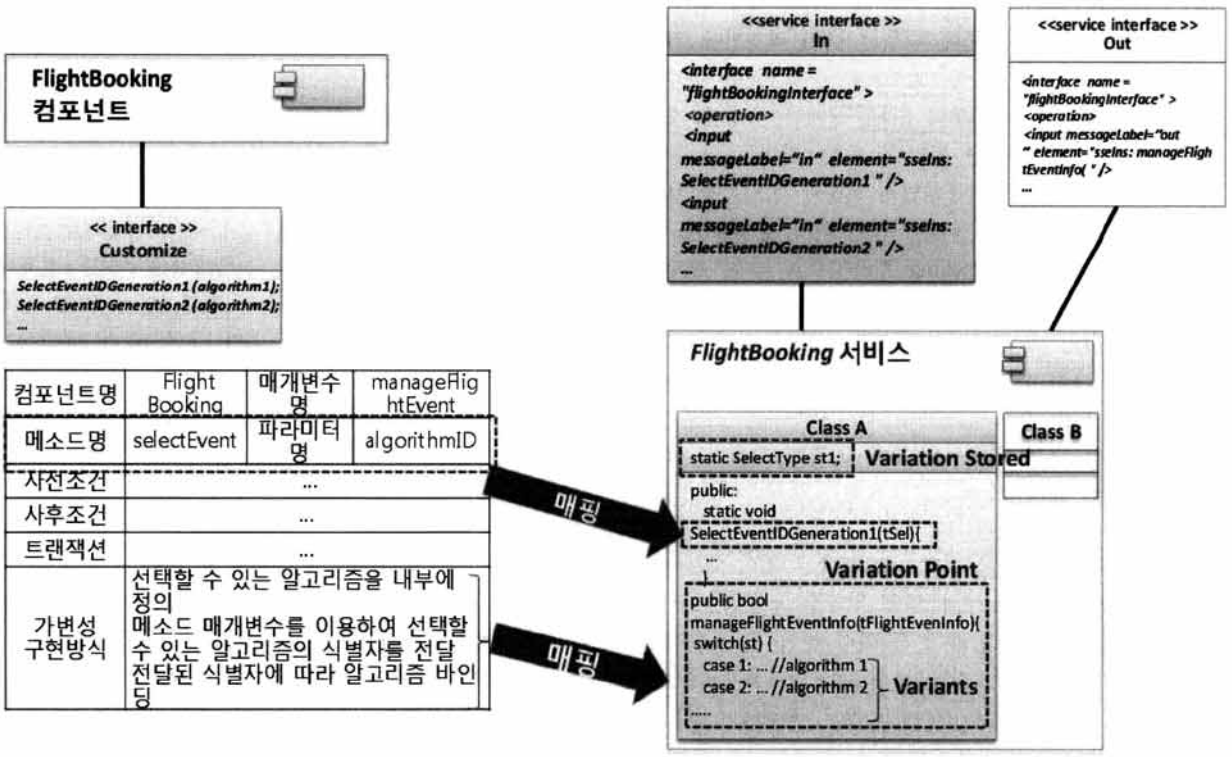
먼저 메소드명과 매개변수 명은 서비스 어댑테이션 모델에서 하나의 서비스 오퍼레이션을 정의하는데 사용된다. 또한, 가변성 구현방식이 실제 가변 가능한 서비스 오퍼레이션을 정의하는 데 사용된다. 이와 같은 정보는 전체매핑 01의 과정을 통해서 얻을 수도 있다.

부분매핑 02. 여행 관리 시스템 비즈니스 로직 컴포넌트와 워크플로우와 부분매핑 05. 여행 관리 시스템 Required 인터페이스와 워크플로우의 경우, 여행 관리 시스템이 특정 문맥에 맞추어진 시스템이어서 복합 서비스로 매핑할 대상이 없기 때문에, 이번 사례 연구에서 다루지 않았다. 또한 부분매핑 04. Required 인터페이스에서 비즈니스 프로세스로의 변환의 경우, 컴포넌트 설계 시 Required 인터페이스에 해당하는 부분은 없었기 때문에 이번 사례 연구에서 다루지 않았으나, 부분매핑 04의 경우 (그림 9)를 통해 충분히 적용성을 보여주고 있다.

6.3 사례 연구 결과 분석 및 평가

본 사례 연구를 통하여 우리는 기존에 있는 여행 관리 시스템을 위한 CBD 설계를 변환하여 SOA 설계를 얻을 수 있었다. 사례연구에서는 논문에서 제안한 9개의 매핑관계 중 6개의 매핑관계를 활용하였고. 이를 기반으로 변환 기법을 적용하였으며, 우리는 다음과 같은 교훈을 얻을 수 있었다.

- CBD 설계와 SOA 설계로의 변환 기법의 효율성: CBD 설계의 결과 산출물의 대부분이 SOA 설계로 변환됨을 확인할 수 있었다. CBD 설계 결과 산출물의 대표적인 것이 컴포넌트 설계서, 컴포넌트 인터페이스 명세서, 공통성 및 가변성 모델이다. 또한 SOA 설계 결과 산출물의 대표



(그림 18) Customize 인터페이스에서 서비스 어댑테이션 모델로의 변환

적인 것이 서비스 설계서, 서비스 인터페이스 명세서, 비즈니스 프로세스 명세서, 서비스 어댑테이션 모델이다.

먼저, 서비스의 경우는 전체매핑 02와 부분매핑 03을 통하여 컴포넌트 설계서로부터 유도할 수 있었다. 특히, 이미 설계되어 있는 클래스 다이어그램을 가져올 수 있었으며, 이에 따라 CBD 설계를 기반으로 구현된 클래스 구현도 가져올 수 있는 기틀을 마련할 수 있었다.

그리고, 서비스 인터페이스 명세서의 경우, 전체매핑 03을 통하여 컴포넌트 인터페이스 명세서에 기술되어 있는 정보를 활용하여 유도할 수 있었다. 특히 서비스 인터페이스의 경우 서비스가 제공하는 기능만을 기술하고 있기 때문에 컴포넌트의 Provided 인터페이스와 쉽게 매핑될 수 있었다. 즉, 컴포넌트의 Provided 인터페이스 명세서에 있는 내용을 서비스 인터페이스 양식으로만 변환하면 되었고 기타 추가작업은 없었다.

비즈니스 프로세스 명세서의 경우, 부분매핑 01을 통하여 비즈니스 로직 컴포넌트에 설계되어 있는 워크플로우 정보를 활용하여 유도할 수 있었다. 전체매핑 02로 인하여 도출된 추가적인 서비스와 관련된 제어흐름을 제외하고는 워크플로우에 기술하였던 대부분의 제어흐름을 그대로 가져올 수 있었다.

서비스 어댑테이션 모델의 경우, 전체매핑 01과 부분매핑 06을 통하여 유도할 수 있었다. 특히 서비스 어댑테이션 모델의 경우 전체매핑 01을 통하여 여행 관리 시스템에서 사용되는 서비스들의 가변성, 가변치 등을 모두 가져올 수 있었기 때문에 설계 측면에서 재사용성이 높다고 할 수 있었다.

SOA 설계에 필요한 정보 및 설계 관련 내용이 이미 CBD 설계에 있었기 때문에 대부분 변환 과정만을 거쳐서 재사용할 수 있었다. 즉, 재사용을 통해 SOA 설계를 수행하였기 때문에, 처음부터 SOA 설계를 수행한 것에 비해 생산성이 향상될 수 있었다. 또한, 이미 운영 중인 시스템의 CBD 설계를 활용하였기 때문에 신뢰성을 보장할 수 있었으며, 설계의 상세도 측면 역시 보장할 수 있었다. 예를 들어 FlightBooking 컴포넌트의 경우, 설계와 설계를 기반으로 한 구현을 가져와서 서비스 설계 및 구현으로 변환하는 데 변환하여야 하는 클래스 설계가 1개, 관련 구현의 LOC 변경이 100 이하였으며, 이를 수행하는 데에 1 시간 이하 소요되었습니다.

- CBD 설계에서 SOA 설계로의 변환 기법의 한계점: 서비스와 컴포넌트의 재사용에 대한 관점이 정확하게 일치하지 않아서 변환에서의 어려움이 발생할 수 있다. Payment와 같은 기능이 서비스로 도출된 것이 그 예이다. 본 논문에서 Payment와 같은 기능을 도출한 이유는 1) 이미 Payment와 관련된 것은 여러 문헌 및 산업계에서 서비스로 제공하고 있고, 2) 서비스는 기본적으로 인터넷을 통한 기능 활용을 가정하고 있기 때문에 CBD 설계와 같이 하나의 기능 모듈에 반드시 포함되어야 하는 것은 아니기 때문이었다. 즉 이처럼 일대 다, 혹은 다대일 매핑이 이루어질 때 관련 매핑에 따른 충분한 근거가 필요하였다.

서비스 인터페이스 명세서의 경우 WSDL에서 요구하는 표준 양식 및 데이터 유형이 있기 때문에 이와 관련된 부분은 변환 기법으로 해결할 수 없었다. 즉, 서비스와 컴포넌트의 사용 및 배치하는 방법이 다르기 때문에 이와 관련한 추가 설계가 필요하였다.

비즈니스 프로세스 명세서의 경우 BPEL에서 요구하는 표준 양식 및 데이터 유형이 있기 때문에 이와 관련된 부분은 변환 기법으로 해결할 수 없었다. 즉, 이에 따른 추가 설계가 필요하였다.

위와 같은 교훈에 따라, 변환 기법의 단점을 보완하기 위하여 우리는 다음과 같이 추가적으로 설계가 필요한 요소를 정리하고 관련 지침을 제공한다. CBD 설계에서 SOA 설계로 변환 시 추가적으로 해야 하는 것은 외부에서 서비스를 호출하기 위한 메시지 전송 프로토콜 정의, 서비스 자체의 엔드포인트 주소 등을 서비스 인터페이스에 추가 명세하는 부분이다. 또한, 복합 서비스나 비즈니스 프로세스의 경우, CBD 설계를 이용하여 워크플로우는 유도할 수 있지만, 복합 서비스 혹은 비즈니스 프로세스 특화된 액티비티는 추가적으로 명세 및 설계해주어야 한다. 마지막으로, BPEL을 이용하여 비즈니스 프로세스를 명세할 경우, 서비스 간의 호출 관계를 표현하기 위하여 BPEL 비즈니스 프로세스가 중간자 역할을 한다. 이에 따라 비즈니스 프로세스 명세 시 서비스와 비즈니스 프로세스 간의 관계를 추가 명세하여야 하며 이때 사용하게 되는 엔드포인트, 메시지 전송 프로토콜 등을 매핑시켜야 한다.

7. 평가 및 결론

컴포넌트 기반 개발과 SOA를 기반으로 한 개발은 방법 측면에서 공통점을 갖고 있다. 즉, 재사용 가능한 기능 단위를 만들고, 이를 기반으로 하여 애플리케이션을 만든다. 그로 인하여, CBD 설계와 SOA 설계 에서 공통점이 있으며, 이런 공통점을 활용하여 본 논문에서는 CBD 설계에서 SOA 설계로의 변환 기법을 정의하였다. 먼저, CBD 설계와 SOA 설계의 핵심요소를 정리하여, 요소 간의 매핑관계를 정의하였다. 그런 다음, 각 매핑에 따른 변환 기법을 지시사항과 예제를 통하여 설명하였다. 마지막으로, 제안한 변환 기법을 활용하여 컴포넌트 기반으로 개발한 여행 관리 시스템의 CBD 설계를 SOA 설계로 변환하였다. 사례연구를 통하여서 보았듯이, CBD 설계에서 얻을 수 있는 산출물 및 요소들은 모두 SOA 설계로 변환할 수 있다. 단, 컴포넌트와 서비스 간의 관계가 다대 다이기 때문에 변환 기법에서 언급하고 있는 고려사항을 충분히 반영하면, 여행 관리 시스템의 CBD 설계를 SOA 설계로 변환한 것처럼 대부분의 CBD 설계의 결과가 SOA 설계로 변환될 수 있을 것이다.

본 논문에서 제안한 변환 기법은 다른 연구와 비교하여 다음과 같은 장점을 갖고 있다.

- 모델 재사용을 통한 생산성 향상: 기존의 Lee의 연구는 코드 재사용을 통하여 CBD 결과물을 활용하는 방안을 제안한다. 그러나 구현 결과를 활용하는 방법의 경우, 특정 플랫폼에 제한되어서 활용될 수 있다. 즉 서비스의 운영 환경이 컴포넌트의 운영 환경과 다른 경우, 구현 결과를 사용하지 못하는 경우가 발생할 수 있다. 또한, Jiang의 연구의 경우 CBD 설계 기술을 SOA 설계에 활용하기 위한 방안을 제안한다. 이 경우 처음부터 SOA를 설계하는 것과 다르지 않은 결과를 가져온다. 그러나 본 논문에서 제안한 변환 기법의 경우 기존에 있는 CBD 설계 모델을 변환함으로써, SOA 설계 자체에 걸리는 소요 시간 및 노력을 줄일 수 있다.
- 모델 간 변환을 활용한 생산성 향상: 기존의 Lee의 연구, Jiang의 연구는 모델 간 변환을 다루고 있지 않다. 모델 간 변환을 통하여 얻을 수 있는 점은 구현과 독립적이기 때문에 실제 변환된 SOA 설계를 여러 플랫폼에서 구현할 수 있다. 그러므로 하나의 설계를 활용하여 여러 플랫폼에서 구현할 수 있으므로 생산성을 향상시킬 수 있다. 본 논문에서 제안한 변환 기법은 모델 간 변환을 통해 플랫폼 독립적인 SOA 설계를 얻을 뿐 아니라, 관련 설계에 따른 구현 결과물도 추가 보충자료로 활용할 수 있는 기틀을 제공하고 있다.
- 상세한 변환 기법을 통한 효율성: 기존의 Brown의 연구는 CBD 설계에서 SOA 설계로 변환하는 데에 추상화 단계를 비즈니스 관점에서 접근한다. 이로 인하여 설계 결과물의 추상화 단계가 높아서, 변환 과정이 이루어진 이후에 추가 설계하여야 하는 요소들이 많이 있다. 그러나 본 논문의 변환 기법의 경우 변환 기법의 추상화 단계를 설계 핵심 요소 관점에서 접근하여서, SOA 특화된 표준에 따른 추가 명세를 제외하고는 추가하여야 하는 작업이 많지 않다. 그러므로 효율성 측면에서 본 논문의 변환 기법이 높다고 할 수 있다.

또한, 이런 변환 기법을 활용하면, 기존에 운영하고 있는 시스템의 CBD 설계를 활용하게 되는 경우가 많기 때문에 신뢰성 있는 SOA 설계를 얻을 수 있다. 향후 연구로는 모델 기반 아키텍처의 기술을 적용하여 CBD 설계의 많은 부분을 SOA 설계로 자동 변환할 수 있는 기법 및 지원 도구의 개발을 수행하여 본 논문에서 제안한 변환 기법의 활용도를 향상시키기 위한 연구를 진행할 예정이다.

참 고 문 헌

[1] Kim, S., "Software Reusability", *in Wiley Encyclopedia of Computer Science and Engineering*, Vol.4, edited by Benjamin W. Wah, pp.2679-2689, Wiley-Interscience, Jan., 2009.

[2] Heineman, G. T., Councill, W. T., *Component-Based Software Engineering*, Addison-Wesley, 2001.

[3] Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, 2005.

[4] Krafzig, D., Banke, K., and Slama, D., *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice Hall, 2004.

[5] 한국전자통신연구원 (ETRI), *마르미-III 버전 4.0*, 2002.

[6] Szyperski, *C., Component Software Beyond Object-Oriented Programming*, Addison-Wesley, 2002.

[7] Arsanjani, A., Zhang, L.J., Ellis, M., Allam, A., and Channabasavaiah, K., "S3: A Service-Oriented Reference Architecture," *IEEE IT Professional Magazine*, Vol.9, No.3, pp.10-17, May/June, 2007.

[8] MacKenzie, C., Laskey, K., McCabe, F., Brown, P., and Metz, R. eds., *Reference Model for Service Oriented Architecture 1.0*, OASIS Standard, 12 October, 2006.

[9] Lee, R., Harikumar, A., Chiang, C., Yang, H., Kim, H, and Kang, B., "A framework for dynamically converting components to Web services", *Third ACIS International Conference on Software Engineering Research, Management and Applications 2005*, pp.431-437, Aug., 2005.

[10] Brown, A., Delbaere, M., Eeles, P., Johnston, S., and Weaver, R., "Realizing Service-oriented Solutions with the IBM Rational Software Development Platform," IBM Systems Journal, Vol.44, No.4, pp.727-752, 2005.

[11] Jiang, M. and Yang, Z., "A Component-Based Service Creation Framework for Mobile Applications," IEEE International Conference on Information Reuse and Integration 2006, pp.307-312, Sep., 2006.

[12] Kim S., Her J., and Chang, S., "A Theoretical Foundation of Variability in Component-Based Development," Information and Software Technology(IST), Vol.47, pp.663-673, July, 2005.

[13] Ferguson, D. and Stockton, M., "Service-oriented architecture: Programming model and product architecture," IBM Systems Journal, Vol.44, No.4, pp.753-780, 2005.

[14] OASIS, *Web Services Business Process Execution Language Version 2.0*, Public Review Draft, 23rd August, 2006.

[15] Havey, M., *Essential Business Process Modeling*, O'Reilly, 2005.

[16] Travelocity Travel: Cheap Airline Tickets, Hotels, Flights, *www.travelocity.com*, Travelocity.

[17] Travel Agency - Liberty Travel, *www.libertytravel.com*, Liberty Travel

[18] Thomas Cook - Book cheap holidays, holiday deals, hotels, flights, *www.thomascook.com*, Thomas Cook

천 두 완

e-mail : raphael.cheun@gmail.com
2005년 숭실대학교 컴퓨터학부(학사)
2006년 숭실대학교 컴퓨터학과(석사)
2006년~현 재 숭실대학교 컴퓨터학과 박사수료
관심분야 : 모바일 컴퓨팅 (Mobile Computing), 서비스 지향 컴퓨팅 (Service Oriented Computing), 서비스 기반 모바일 어플리케이션 관리

조 성 현

e-mail : shjo@nipa.kr
1997년 한국외국어대학교 영어과(학사)
2002년 한국외국어대학교 경영정보대학원 MIS전공(석사)
2009년 한국외국어대학교 대학원 경영학과 (박사수료)
2002년 1월~2004년 1월 대림정보통신
2004년 2월~현 재 정보통신산업진흥원(NIPA) 소프트웨어공학 연구팀 책임연구원
관심분야 : 서비스 지향 컴퓨팅 (Service Oriented Computing), 컴포넌트 기반 설계

김 수 동

e-mail : sdkim777@gmail.com
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~현 재 숭실대학교 컴퓨터학부 교수
관심분야 : 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스(Mobile Service), 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 소프트웨어 아키텍처