CBD의 부분 매칭 문제 해결을 위한 커넥터 설계 기법

(A Method to Design Connectors to Resolve Partial Matching Problems in CBD)

민 현 기 † 김 수 동 ††
(Hyun Gi Min) (Soo Dong Kim)

요 약 효율적 재사용 기술인 컴포넌트 기반 개발(Component-based Development, CBD)이 보급화되고 있다. 특히 Commercial-Off-The-Shelf (COTS) 컴포넌트는 컴포넌트 내부의 구조 보다 컴포넌트 간의 연결 구조가 중요하다. 컴포넌트 사용자가 요구하는 기능과 적절한 인터페이스를 제공하는 컴포넌트를 식별하여 재사용하는 일은 CBD에서 핵심 작업 중 하나이다. 만약 식별된 후보 컴포넌트가 제한된 활용성과 기능을 제공한다면, 컴포넌트 사용자는 어플리케이션을 개발할 때 컴포넌트들을 재사용할 수 없다. 후보 컴포넌트와 사용자가 요구하는 사항과의 불일치를 해결하기 위해 스마트 커넥터가 필요하다. 그러나 기존 연구는 컴포넌트 불일치 해결을 위한 커넥터의 종류는 정의되어 있지만, 컴포넌트간의 부분 매칭을 식별하기 위한 방법과 절차가 부족하다. 본 논문에서는 컴포넌트의 부분 매칭 문제를 식별하기 위한 분류 기법을 제시한다. 이 분류 기법을 이용하여 부분 매칭을 해결하기 위한 스마트 커넥터의 종류를 식별하고, 커넥터를 설계하기 위한 기법을 제안한다. 따라서 부분 매칭 문제를 갖는 컴포넌트도 어플리케이션 개발시 사용될 수 있다.

키워드 : 커넥터, 컴포넌트 기반 개발(CBD), 컴포넌트 식별

Abstract Component-based Development (CBD) is gaining popularity as an effective reuse technology. Especially commercial-off-the-shelf (COTS) components are mainly for inter-organizational reuse, rather than intra-organizational reuse. One of the essential tasks in CBD is to reuse the right components that provide the functionality and interface required by component consumers. If candidate components provide a limited applicability and customizability, a component consumer cannot reuse the components in application development. To resolve this partial matching problem, we need smart connectors that fill the gap between candidate components and the specification of components required. Previous researches about smart connector describe only connector types to resolve mismatch problems. However, previous researches do not address the identification and design method to resolve the problems. In this paper, we suggest taxonomy of various mismatch problems to identify partial match problems between requirements of the application and components. We identify smart connector types and suggest a systematic process for designing smart connectors using the taxonomy. Therefore, components that have the problems can be used to develop applications.

Key words : Connector, Component-based Development (CBD), Component Identification

1. 서 론

효율적인 재사용 기술인 CBD 기법에 의한 어플리케이션 개발이 보급화되었다. 사용자들은 컴포넌트를 획득해서 사용하기 전에 COTS 컴포넌트를 평가하려 한다[1]. COTS 컴포넌트들의 품질을 평가하는 것은 성공적인 컴포넌트 기반의 어플리케이션 개발에 있어서 미리 수행되어야 할 중요한 작업이다[2].

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 정 회 원 : 숭실대학교 정보미디어기술연구소 전임연구원
hgmin@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
논문접수 : 2005년 5월 2일
심사완료 : 2005년 10월 14일

특히 어플리케이션 개발을 위해 적절한 컴포넌트를 획득하는 일이 CBD에서 중요하지만, 사용자가 요구하는 모든 기능을 제공하는 COTS 컴포넌트를 획득하기는 어렵다. 선택된 후보 컴포넌트들이 사용자에 의해 요구되는 기능의 일부분이 적합하지 않은 경우가 있다. 따라서 개발자들은 만족스러운 컴포넌트들을 찾기 위해 많은 시간을 보내고 있다[3]. 그러나 오랜 노력에 의해 선택된 컴포넌트들도 부분 일치 문제가 생긴다. 이런 경우에 컴포넌트 사용자는 컴포넌트를 사용할 수 없다. 이러한 후보 컴포넌트와 사용자에게 요구되는 컴포넌트간의 차이점을 극복하기 위해 커넥터가 필요하다[4]. 그러나, 커넥터는 모듈간의 연결만을 명세한 개념적인 수준에 머물러 있고, 이러한 커넥터를 실용적으로 사용 할 수 있게 커넥터를 분석, 설계 후 구현까지 지원하는 연구는 부족하다.

본 연구에서는 컴포넌트간의 부분 매칭 문제(Partial Matching Problem)로 사용할 수 없는 컴포넌트를 재사용하기 위한 커넥터의 설계 프로세스와 지침을 제안한다. 커넥터의 기능성을 4종류로 정의하고, 컴포넌트간 부분 매칭 문제가 일어나는 다양한 상황과 이를 해결하기 위한 18개의 관련 행위로 분류한다. 제안한 분류법에 의해 컴포넌트의 부분 매칭을 식별한다. 이러한 부분 매칭을 해결하기 위한 커넥터를 설계하기 위해 4개의 단계와 세부 활동과 지침을 제안한다.

본 논문의 구성은 다음과 같다. 2장은 컴포넌트 프레임워크, 커넥터, 컴포넌트 획득 프로세스에 관한 기반 연구를 기술한다. 3장은 스마트 커넥터에 관해 기술한다. 4장은 부분 매칭이 발생되는 경우를 설명한다. 5장은 커넥터를 설계하기 위한 프로세스를 4단계로 제안한다. 6장은 커넥터 설계의 사례연구를 보인다. 7장과 8장은 본 논문을 평가하고 결론을 내린다.

2. 기반 연구

2.1 컴포넌트 프레임워크

컴포넌트 프레임워크는 컴포넌트, 관련된 컴포넌트들의 연관 관계, 커넥터, 제약사항들의 집합인 대형의 재사용 단위이다. 컴포넌트 프레임워크는 컴포넌트보다 재사용 단위가 더 크다. 어플리케이션 개발 시 모든 컴포넌트들을 조립하지 않고, 미완성된 어플리케이션(Semi-application)인 컴포넌트 프레임워크에 필요한 컴포넌트들을 추가하여 구축한다. 컴포넌트 프레임워크에서 컴포넌트들 사이의 논리적인 연결들은 결합도를 낮춰야 한다. 컴포넌트 수정 없이 재설치 할 수 있는 컴포넌트의 프레임워크를 만들기 위해 컴포넌트들간의 상호작용과 연관관계를 체계적으로 설계해야 한다[4].

2.2 커넥터

커넥터는 컴포넌트 프레임워크 기반에서의 컴포넌트간의 연결에 사용되며, 컴포넌트들간의 상호작용을 표현할 수 있다. 커넥터는 커넥터가 연결되는 포트에 역할 및 특정한 제약 사항을 부과하며, 컴포넌트간의 연결행위(Joint Action)를 구현하는 특정한 상호작용을 위한 프로토콜로 정제된다[5]. 상호작용을 설계하기 위해 커넥터를 사용하는 것은 교체단위가 있는 컴포넌트 프레임워크 설계를 위해 유용한 기술이다. 어플리케이션을 만들기 위해 부분 매칭 문제가 있는 컴포넌트는 그림 1과 같이 커넥터를 사용하여 어플리케이션 개발시 사용될 수 있다.

커넥터는 입력 포트(Input port)과 출력 포트(Output port)를 가지고 있는 작은 소프트웨어 부품이다. 이러한 포트는 컴포넌트와 연결되기 위해 사용된다[6]. 그림 1처럼 *CLIENT*(Source Component)는 커넥터를 통해 *COMP*(Target Component)에 메시지를 전달한다. 이러한 메시지는 *CLIENT*의 Required 인터페이스에서 커넥터의 입력 포트를 통해 전달되고, 커넥터는 출력 포트를 통해 타겟 컴포넌트와 연결된다[7].

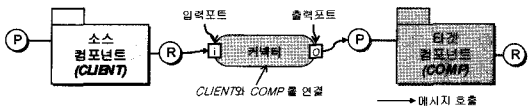


그림 1 컴포넌트와 커넥터의 상호 관계

2.3 PORE 프로세스

획득 기반의 요구 공학(Procurement-Oriented Requirements Engineering, PORE)은 요구사항에 만족하는 COTS 컴포넌트를 획득하는 방법을 제공한다[4]. 그림 2와 같이 PORE는 요구사항 식별과 COTS 소프트웨어 획득을 반복적으로 수행하면서 적절한 COTS 소프트웨어를 찾는다. 결정 기준에 의해 요구사항을 만족하는 후보 컴포넌트들이 식별된다. 이러한 요구사항을 맞지 않는 COTS 컴포넌트는 제거된다. 이러한 과정을 반복 수행하여 남게 되는 후보 컴포넌트가 사용 가능한 COTS 컴포넌트이다.

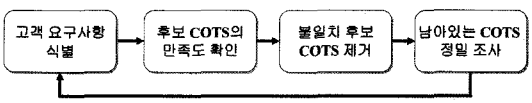


그림 2 COTS 소프트웨어를 획득하기 위한 프로세스

3. 스마트 커넥터(Smart Connector)

COTS 컴포넌트는 일반적으로 바이너리의 블랙박스 형식으로 제공된다. 그러므로 컴포넌트 사용자들은 컴포

넌트의 소스 코드를 수정할 수 없다. 만약 후보 컴포넌트가 고객이 원하는 기능성과 인터페이스를 제공하지 못한다면, 그 컴포넌트 사용자는 어플리케이션 개발할 때 해당 컴포넌트를 사용할 수 없다. 우리는 이러한 문제를 컴포넌트 획득에서의 부분 매칭 문제라 부른다[8].

본 논문의 커넥터는 소프트웨어 아키텍처의 모듈간의 연결을 명세하기 위한 커넥터가 아닌, 컴포넌트와 컴포넌트 사용자 사이에서 연결되는 바이너리 형식의 소프트웨어 모듈이다. 본 논문의 커넥터는 부분 매칭 문제를 해결 하기 위한 변환 규칙을 가지고 있으므로, 소프트웨어 아키텍처에서 연결을 의미하는 커넥터와 구별하기 위해 본 논문에서는 스마트 커넥터라 명한다. 본 논문에서는 그림 3과 같이 사용자를 *CLIENT*라고 정의하고, 사용되는 컴포넌트를 *COMP*라 정의한다. *CLIENT*는 컴포넌트 또는 어플리케이션 프로그램도 될 수 있다.

3.1 인터페이스 어뎁터(Interface Adapter)

인터페이스 어뎁터(Interface Adapter)는 *CLIENT*에서 요구하는 인터페이스의 시그니처와 *COMP*에서 제공하는 시그니처간의 부분 매칭 문제를 해결하기 위해 사용된다. 즉, 인터페이스의 시그니처가 서로 달라 사용못하는 컴포넌트들을 사용 가능 하도록 한다. *CLIENT*는 인터페이스 어뎁터에 *COMP*와 구문적 부분 매칭이 되는 메시지를 전달하고, 인터페이스 어뎁터는 변환 규칙에 따라 부분 매칭 메시지를 *COMP*에서 제공하는 인터페이스에 맞게 변환시킨다. 인터페이스 어뎁터는 재정의된 메시지를 *COMP*로 전달한다. 또한 인터페이스 어뎁터는 반환값의 데이타형의 부분 매칭도 변환하여 *CLIENT*에 그림 3과 같이 전달한다.

3.2 데이타 변환자(Value Range Transformer)

데이타 변환자(Value Range Transformer)는 *CLIENT*에 의해 요구되는 데이타의 의미(Semantic)와 *COMP*에서 사용되는 데이타간의 의미를 보정해주는 역할을 한다. 데이타 변환자는 *CLIENT*로부터 데이타를 전달 받은 후 그 데이타의 의미를 *COMP*에서 요구하는 의미로 변환 규칙에 의해 변환한 후 *COMP*로 전달한다. 예로 *CLIENT*와 *COMP*에서 사용되는 온도 데이타형은 모두 float형이지만, *CLIENT*에서는 섭씨 온도를 전달하고, *COMP*에서는 화씨 온도를 사용하는 경우, 컴파일 에러는 발생되지 않는다. 하지만 이러한 데이타의 의미의 차이는 시스템에 심각한 문제를 발생시킨다. 유사한 경우로 *COMP*에서 반환한 데이타형과 *CLIENT*가 원하는 반환값의 데이타형이 같지만, 그 의미가 다른 경우에서 커넥터에서 변환 규칙에 의해 변환 후 반환이 가능하다.

3.3 기능 변환자(Functional Transformer)

기능 변환자(Functional Transfer)는 *CLIENT*에서 기대하는 기능과 COMP에서 제공하는 기능간의 부분 매칭을 해소하기 위한 기법을 제공한다. 획득된 COTS 컴포넌트들은 대부분의 기능이 만족되기 때문에 컴포넌트 사용자에 의해 획득되었을 것이다. 하지만 모든 기능을 만족시키는 COTS 컴포넌트를 획득하기는 어렵다. *CLIENT*에서 요구하는 기능과 *COMP*에서 제공하는 기능간의 관계는 3종류로 분류 할 수 있다.

컴포넌트의 기능성은 컴포넌트에서 제공하는 모든 기능들의 집합으로 정의된다. 그림 4에서 *Fn(COMP)*는 획득한 컴포넌트에서 제공하는 기능을 의미하고, *Fn*

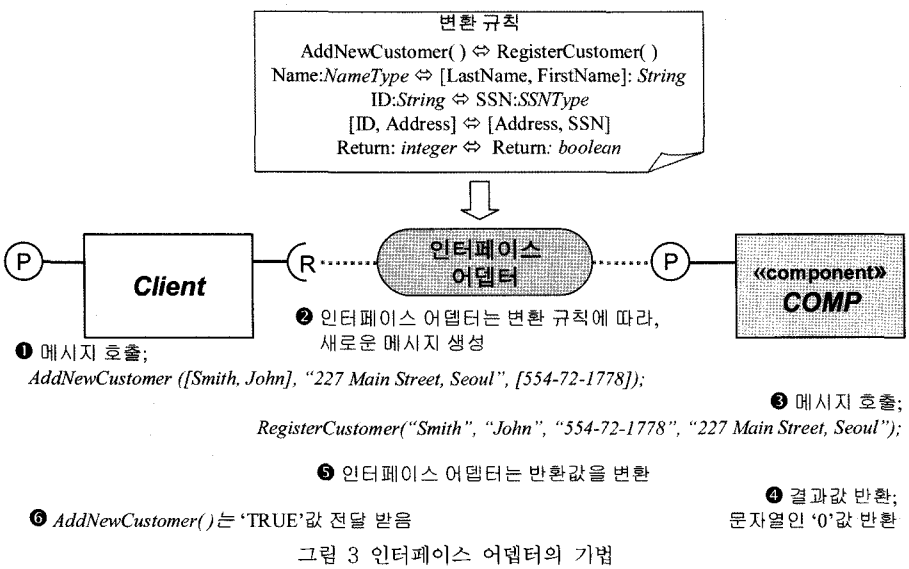


그림 3 인터페이스 어뎁터의 기법

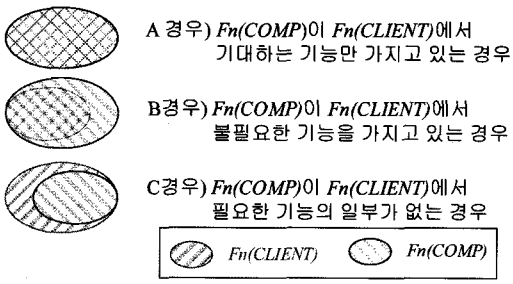


그림 4 사용자 요구사항과 컴포넌트 기능성과의 관계

*(CLIENT)*는 *COMP*사용자가 기대하는 기능을 의미한다. (A)의 경우에는 사용자가 요구하는 기능을 *COMP*에서 완벽하게 지원하는 경우이다. (B)의 경우에는 사용자가 요구하는 기능을 COTS 컴포넌트에서 모두 제공을 하고 있지만, 불필요한 기능도 가지고 있다. 이러한 불필요한 기능이 시스템의 성능을 떨어트리고, 원치 않는 작업으로 인해 다른 기능에 영향을 줄 수 있다. (C)의 경우에는 사용자가 원하는 기능의 일부를 COTS 컴포넌트에서 제공하고 있지 않은 경우이다. 기능 변환자는 부족한 기능을 제공한다. 또한 *COMP*에서 불필요한 기능을 보정하는 기능이 있다면 커넥터에서 보정하기 위한 기능을 제공한다.

3.4 워크플로우 핸들러(Workflow Handler)

워크플로우 핸들러는 요구사항에서 원하는 메시지 호출 순서인 워크플로우와 컴포넌트가 제공하는 워크플로우가 일치하지 않는 경우에 이를 보정한다. 기능 변환자의 사용은 기능의 지원 여부에 따라 사용되고, 워크플로우 핸들러는 인터페이스에서 제공하는 오퍼레이션들의 호출 순서의 부분 매칭 문제 해결을 위해 사용된다.

워크플로우 핸들러는 인터페이스 안에 있는 기능을 수행하기 위해 컴포넌트들간 메소드 호출의 순서를 관리한다. 워크플로우 핸들러는 그림 5와 같이 *CLIENT*에서 제공하는 워크플로우를 요구사항에서 기대하는 워크플로우로 변경이 가능하다. 워크플로우 핸들러는 새로운 메시지 호출을 수행하거나, *CLIENT*로부터 전달 받은 메시지를 보관(Hold)하며, 적절한 메시지들로 조합하여 보관된 메시지를 *COMP*로 전달하는 역할을 수행한다. 따라서, 해당 컴포넌트들에는 영향을 주지 않고, 커넥터에서 그 워크플로우를 조절을 한다.

4. 컴포넌트의 부분 매칭 식별 기법

커넥터에 관한 기존 연구들은 단지 컴포넌트 모듈간의 연결만을 설명하고 있다. 따라서 요구사항 명세서와 컴포넌트간의 부분 매칭, 그리고 컴포넌트와 컴포넌트간의 부분 매칭이 일어나는 상황에 대한 연구가 필요하다.

컴포넌트간의 불일치가 발생되는 곳과 가변성이 일어나는 곳은 컴포넌트의 참조 모델에서 유도 될 수 있다[9]. 컴포넌트의 참조 모델은 그림 6과 같다. 컴포넌트의 참조 모델은 클래스, 인터페이스, 클래스들간의 워크플로우로 구성된다. 클래스는 속성과 메소드를 가지고 있다. 어떤 클래스들이 가지고 있는 속성들은 영속적인 저장 매체에서 저장되고 관리되어야 한다. 컴포넌트 참조 모델의 구성 요소는 속성, 로직, 워크플로우, 영속성, 인터페이스로 구성된다[9].

컴포넌트 참조 모델을 기반으로 한 컴포넌트의 부분

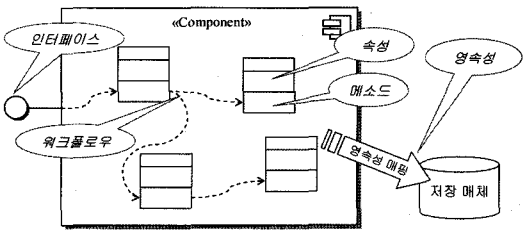


그림 6 컴포넌트의 참조 모델

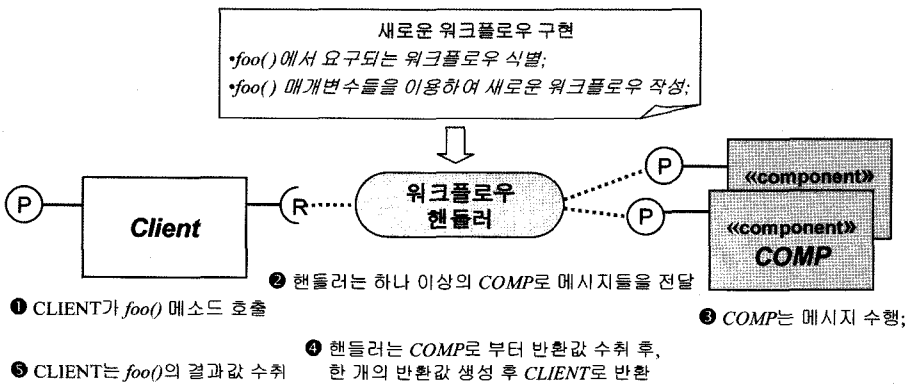


그림 5 워크플로우 핸들러의 기법

매칭의 분류 체계는 표 1과 같이 5종의 큰 분류와 18종의 작은 분류로 나누어진다. 제안한 부분 매칭의 분류 기준들을 이용하여 컴포넌트간의 부분 매칭을 식별하여, 커넥터 구현을 위한 커넥터 요구사항 명세서를 작성할 때 사용된다. 컴포넌트의 부분 매칭 분류는 부분 매칭 문제가 발생되는 상황과 이들을 해결하기 위해 요구되는 해결 방법은 표 1과 같다.

컴포넌트의 부분적 매칭 문제가 발생되는 상황을 인터페이스의 시그니처, 데이타의 의미, 기능성, 워크플로우, 영속성 5가지로 분류한다. 인터페이스 부분 매칭 중 매개변수 데이타형의 부분 매칭은 *CLIENT*에서 전달한 매개변수의 정보와 *COMP*가 기대하는 정보가 일치하지만, 전달 받는 매개변수의 데이타형이 서로 부분 매칭하여 *COMP*에서 전달 받을 수 없는 상황이다. 이때는 데이타형이 틀린 매개변수를 보정한 후 *COMP*로 전달한다. 예외처리 방법의 부분 매칭은 어떤 *CLIENT*는 업무 수행 중 오류가 발생되면, boolean형의 반환값을 받거나, Java의 Exception 객체의 Throw에 의해 해결하는 경우가 있다. Exception 객체를 Throw 하는 함수를 호출하는 *CLIENT*는 try-catch문을 사용해야 한다. 그러므로 *COMP*의 예외 발생 처리 방법에 따라 *CLIENT*의 코드가 변경되어야 하므로 이를 방지하기 위해 예외처리 기법을 제공해야 한다.

표 1 컴포넌트의 부분 매칭 분류

구분	부분 매칭 발생 상황
인터페이스	I1.오퍼레이션의 이름 불일치 예) addItem( ) → createItem( )
	I2.매개변수의 데이타형 불일치 예) String → long
	I3.매개변수들의 순서 불일치 예) Fn(name:String, age : int) → Fn(age:int, name:String, phone:String)
	I4.반환값의 데이타형 불일치
	I5.예외 처리 방법 불일치 예) false값 반환 Exception("오류") 객체 전달
데이타 의미	D1.매개변수의 데이타 의미 불일치 예) 100°C → 212°F
	D2.반환값의 데이타 의미 불일치 예) "Y" or "N" → "Ok", "Fail'
기능성	F1.부족한 기능성 제공
	F2.불필요한 기능성 제공
	F3.부족한 기능성 및 불필요한 기능성 제공
	F4.불필요한 기능성 제거시 부작용 발생
워크플로우	W1.새로운 워크플로우 필요 예) Fn1( ), Fn3( ) → Fn1( ), Fn2( ), Fn3( ),
	W2.워크플로우의 메시지 호출순서 불일치 예) Fn1( ), Fn2( ) → Fn2( ), Fn1( )
	W3.호출되는 메시지의 부분적 통합 예) 입금( ), 출금( ) → 계좌이체 ( )
	W4.호출한 메시지가 여러 개로 분리 Ex) 계좌이체( ) → 입금(), 출금()
	W5.워크플로우 변경에 따른 부작용 발생
영속성	P1. 영속성을 위한 저장 매체의 불일치 Ex) 관계형 데이타베이스, 객체지향 데이타베이스, 파일 시스템
	P2. 저장 스키마의 불일치 Ex) 상속관계의 Unification, Horizontal, Vertical 분할

데이타 의미의 부분 매칭 발생은 데이타의 구문적 에러가 *CLIENT*와 *COMP*간에 발생되지는 않지만, 그 매개변수 및 반환 데이타의 의미가 달라서 발생되는 경우이다. 가령, *CLIENT*와 *COMP*가 서로 다른 도량형을 쓰는 경우에 발생된다. *CLIENT*의 10은 10미터(m)를 의미하고, *COMP*에서는 10을 10마일(mile)로 의미할 수 있다. 컴포넌트의 기능 부분 매칭은 3.3절의 기능 변환자에서 서술한 것처럼 기능이 부족하거나 불필요한 기능을 내포 할 수 있다.

컴포넌트의 워크플로우 부분 매칭은 다른 부분 매칭 문제보다 복잡하다. 단순히 새로운 메시지만을 추가로 호출하는 경우도 있지만, 일반적으로 컴포넌트의 오퍼레이션들이 이미 호출되는 순서를 예상하고 구현된 경우에는 좀 더 복잡하다. 이러한 경우에는 호출 순서의 변경에 따른 부작용이 발생 될 수 있으므로, 이 부작용을 해결하기 위한 보정 기능이 추가가 되어야 한다. 가령, 전자상거래의 주문 정보에 의해 배송과 결재업무가 수행될 때 결재업무가 완료되어야 배송 정보를 등록할 수 있도록 개발된 배송 컴포넌트가 있을 때, 결재업무와 배송업무의 순서를 변경하여 구현하기 위해서는 이러한 종속관계를 해결하기 위한 업무가 요구된다.

영속성에서 데이타베이스의 스키마는 속성을 저장하기 위해 컴포넌트에 구현된 기법에 종속적으로 구축될 것이다. 하지만 컴포넌트를 획득한 사용자측에서 이미 고정시킨 데이타베이스가 있고, 컴포넌트에서 구현한 스키마와 이미 가지고 있는 데이타베이스 스키마와 맞지 않는 경우에 영속성 불일치 문제가 발생된다. 하지만 컴포넌트의 저장 기법과 데이타베이스 스키마가 다른 문제를 컴포넌트간의 불일치를 해결하기 위한 커넥터에서 해결하기는 어렵다. 따라서 본 논문의 커넥터는 인터페이스, 데이타 의미, 기능성, 워크플로우의 부분 매칭 문제를 해결한다.

5. 커넥터 개발 프로세스

본 장에서는 커넥터를 설계하고 구현하기 위한 프로세스 및 지침을 제안한다. 커넥터를 개발하기 위한 단계는 그림 7과 같이 후보 컴포넌트 획득, 컴포넌트 부분 매칭 식별, 커넥터 설계, 컴포넌트와 커넥터 구현으로

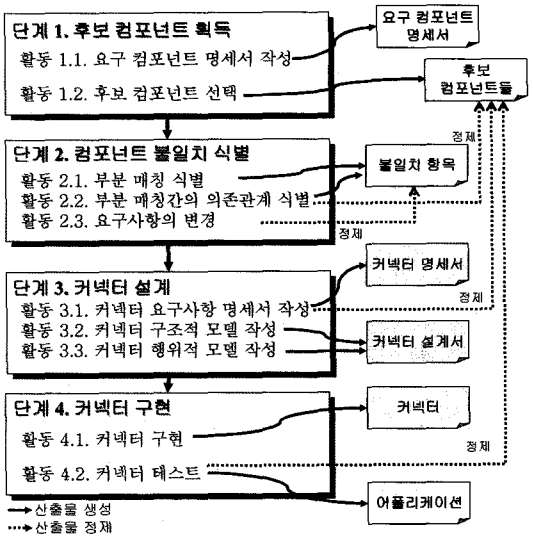


그림 7 커넥터 개발 프로세스 및 산출물 관계

이루어진다.

5.1 단계 1. 후보 컴포넌트 획득

단계 1은 어플리케이션을 조립하기 위해 요구하는 COTS 컴포넌트를 명세하고, 사용할 COTS 컴포넌트를 검색하고, 그 중 적절한 컴포넌트를 선택하여 획득하는 단계이다.

활동 1.1 요구 컴포넌트 명세서 작성. 타겟 시스템에 적절한 컴포넌트를 획득하기 위해 어플리케이션 개발자가 요구하는 컴포넌트가 무엇인지를 알아야 한다. 따라서 그들의 요구사항을 작성한 컴포넌트의 요구사항 명세서가 필요하다. 또한 기존에 가지고 있는 컴포넌트와 조립되기 위해 기존 컴포넌트가 희망하는 컴포넌트의 요구사항 명세서도 필요하다. 요구 컴포넌트 명세서의 예는 그림 8과 같다. 요구 컴포넌트 명세서는 희망하는 기능성, 희망하는 컴포넌트의 인터페이스, 관리되어야 하는 데이타, 컴포넌트의 제약사항 등으로 구성된다.

요구 컴포넌트 명세서

1. 컴포넌트 이름 : Membership Manager
2. 기능성
   - 사용자 등록 : 새로운 사용자의 정보를 영구적으로 등록한다.
   - 사용자 수정 : 이미 등록된 사용자의 정보를 수정한다.
   - ...
3. 요구되는 인터페이스
   - registerMember (id : String, pw : integer, ...) : void
   - ...
4. 관리되는 데이터
   - 사용자 아이디, 패스워드, 이름, 생년월일, 성별, 신용등급
   - ...
5. 제약 사항
   - EJB 2.0 기반의 컴포넌트
   - BEA WebLogic 8.0 이상

그림 8 요구되는 컴포넌트 명세서의 예

컴포넌트 제약사항은 컴포넌트가 구현된 플랫폼 언어, 미들웨어, 품질 속성(Quality Attribute)등의 획득될 컴포넌트의 제약사항을 명세한다.

활동 1.2 후보 컴포넌트 선택. 어플리케이션 개발자는 컴포넌트 시장에서 타겟 시스템의 도메인을 위한 컴포넌트를 검색하여 후보 컴포넌트 리스트를 작성한다. 그리고, 후보 컴포넌트의 컴포넌트 명세서가 함께 제공되는지를 확인한다. 컴포넌트 명세서는 컴포넌트 API 설명, 컴포넌트의 제약사항, 컴포넌트 커스터마이제이션 기법 등에 대한 설명서이다.

작성된 요구 컴포넌트 명세서와 후보 컴포넌트의 명세서의 기능적, 비기능적 요구사항[10]에 위배되는지 확인한다. 가령, 타겟 어플리케이션을 위한 컴포넌트 플랫폼을 사용하였는가? 해당 컴포넌트를 사용하기 위한 주변 자원 및 환경이 적절한가? 컴포넌트의 처리 성능을 만족하는가?에 대한 사항들을 확인한다. 이러한 과정을 통해 부적절한 여러 후보 컴포넌트들이 탈락하게 된다. COTS 컴포넌트의 사용이 부분적으로 허용된 경우에는 테스트가 가능할 것이다. 따라서 컴포넌트 명세서와 컴포넌트가 개략적으로 일치하는지 조사한다. 하지만, COTS 컴포넌트를 구매 전에 해당 컴포넌트의 사용이 제약된다면, 구체적으로 컴포넌트 명세서와 일치하는지 면밀이 검사하기는 어렵다.

5.2 단계 2. 컴포넌트 부분 매칭 식별

단계 2는 타겟 시스템의 요구사항 명세서와 컴포넌트의 부분 매칭 문제점 식별 및 컴포넌트간의 의존관계의 부분 매칭 문제점을 식별하는 단계이다.

활동 2.1 컴포넌트의 부분 매칭 식별. 이 활동에서는 컴포넌트의 부분 매칭 문제를 찾는다. COTS 컴포넌트가 컴포넌트 획득 과정을 통해 선택되었지만, 상세하게 조사 되어 선택된 COTS 컴포넌트가 아니다. 따라서 *COMP*와 *CLIENT*간에는 부분 매칭 문제를 가지고 있다. 부분 매칭 문제는 2가지 종류로 분류된다. 첫째, 시스템의 요구사항을 획득한 컴포넌트가 만족시키지 못하는 경우가 있고, 컴포넌트들을 조립하기 위해 컴포넌트와 컴포넌트간의 부분 매칭 문제가 발생하는 경우가 있다. 이 활동에서는 고객이 요구하는 사항을 컴포넌트가 만족시키는 사항들을 체크하게 된다.

요구 사항 명세서를 기반으로 유스 케이스 모델을 작성된다. 유스 케이스 모델은 유스 케이스의 기능, 주요 흐름, 대안 흐름, 예외 흐름, 시나리오 등을 명세한다. 그리고 그림 9와 같이 선택된 유스 케이스와 컴포넌트를 비교한다. 이때 비교하는 항목은 유스 케이스의 기능, 해당 유스 케이스를 수행할 때 관리되어야 할 데이타, 유스 케이스의 흐름이다. 컴포넌트가 사용자의 요구사항을 만족시키는 일치하는 항목과 만족시키지 못하는

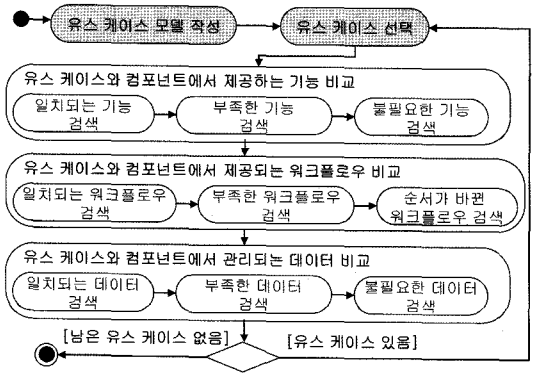


그림 9 유스 케이스 기반의 부분 일치 문제 식별

부분 일치 등을 식별한다. 컴포넌트 인터페이스의 시그니처 불일치는 후보 컴포넌트 획득시 파악 된다.

활동 2.2 부분 매칭들간의 의존 관계 식별. 이 활동에서는 식별된 부분 매칭간의 의존 관계를 식별한다. 활동 2.1에서 발견된 부분 매칭 중 해당 내용이 변경되면, 다른 컴포넌트에 그 영향이 파급되는지를 식별한다. 또한 워크플로우 관점에서 컴포넌트의 오퍼레이션 호출 순서의 변경이 다른 오퍼레이션에 영향을 미치는지 조사한다. 이와 같이 식별된 부분 매칭간의 의존 관계는 컴포넌트의 기능성과 워크플로우에서 발견될 수 있다. 의존 관계의 보정이 다른 의존 관계의 문제에 서로 영향을 준다면, 보정이 불가능하여 조립이 불가능하므로 해당 컴포넌트는 후보 컴포넌트에서 탈락된다.

활동 2.3 요구사항의 변경. 이 활동에서는 컴포넌트의 부분 매칭 문제를 해결하기 위해 요구사항을 변경하는 활동이다. 활동 2.1과 활동 2.2에서 식별된 부분 매칭 문제로 인하여 COTS 컴포넌트를 사용하지 못할 경우에는 원하는 어플리케이션에 크게 영향을 미치지 않는 수준에서 요구사항을 컴포넌트에 맞도록 변경한다. 현실적으로 사용자의 모든 요구사항을 만족시키는 컴포넌트를 획득하기는 어려우므로, 적정 수준에서 요구사항을 변경함으로 부분 일치 문제를 가지고 있는 컴포넌트도 사용할 수 있게 된다.

5.3 단계 3. 스마트 커넥터 설계

단계 3은 상위 단계에서 식별한 후보 컴포넌트의 부분 매칭 문제를 해결하기 위한 커넥터를 설계한다. 커넥터 요구사항 명세서 작성, 커넥터의 구조 설계, 커넥터의 행위 설계로 이루어진다.

활동 3.1 커넥터 요구사항 명세서 작성. 이 활동의 목적은 컴포넌트의 부분 매칭 항목을 작성하고, 이 부분 매칭을 해결하기 위한 커넥터의 요구사항을 작성한다. 커넥터의 요구사항 명세서의 항목은 표 2와 같다. 첫째, 컴포넌트 부분 매칭과 관련된 *CLIENT*의 Provided 인터페이스를 간략하게 명세 한다. 이는 COMP가 수행되는 배경을 설계자에게 제시한다. 그리고 이 Provided 인터페이스가 수행되는 동안 호출하는 Required 인터페이스를 작성한다. 즉 COMP가 이것을 충족시키지 못해서 부분 매칭이 일어난 것이다.

둘째, 부분 매칭 문제가 있는 해당 후보 컴포넌트의 Provided 인터페이스를 작성한다. 이때는 커넥터 설계자가 명세된 Provided 인터페이스를 검토하고, *CLIENT*에서 요구하는 사항이 무엇인지 파악할 수 있도록 Provided 인터페이스의 명세서의 내용을 그대로 옮기는 것보다는 *CLIENT*와 부분 매칭 문제 관점에서 다시 작성하는 것이 효율적이다. 셋째, 해결해야 할 부분 매칭 문제를 작성한다. *CLIENT*의 Required 인터페이스와 COMP의 Provided 인터페이스 명세를 고려하여 해결을 위한 초안을 작성한다.

넷째, 작성된 부분 매칭 문제를 해결하기 위한 커넥터

표 2 커넥터 요구사항 명세서의 항목

대 항 목	소 항 목	내 용
CLIENT : *<컴포넌트 식별자 >*	Provided 인터페이스	*<Provided 인터페이스 서술>*
	Required 인터페이스	*<Required 인터페이스 서술>*
	비 고	*<부분 매칭 문제 관점에서 추가 서술>*
COMP : *<컴포넌트 식별자>*	Provide 인터페이스	*<Provided 인터페이스 서술 >*
	비 고	*<부분 매칭 문제 관점에서 추가 서술>*
부분 매칭 문제	문 제 점	*<부분 매칭 문제가 일어나는 상황 서술>*
Connector: *<커넥터 식별자>*	커넥터 분류자	*<부분 매칭 해결을 위한 커넥터의 분류자 서술 >*
	해 결 방 법	*<커넥터가 수행할 알고리즘 서술>*
	포 트	*<커넥터 포트 서술>*
	불 변 조 건	*<커넥터의 불변 조건 서술>*
	사 전 조 건	*<커넥터의 사전 조건 서술>*
	사 후 조 건	*<커넥터의 사후 조건 서술>*
테스트 케이스	입 력 값	*<테스트를 위한 입력값>*
	출 력 값	*<테스트를 위해 예상된 결과 >*

의 이름을 정한다. 해당 커넥터의 역할이 잘 나타나는 이름으로 한다. 그리고 표 1의 부분 매칭 상황과 해결 방법을 이용하여 인터페이스 어뎁터, 데이타 변환자, 기능 변환자, 워크플로우 핸들러 등의 커넥터의 패턴을 기입한다. 데이타형도 다르고, 데이타의 의미도 다른 경우처럼, 부분 매칭이 복합적인 경우에는 2가지 이상의 이름도 가능하다.

다섯째, 부분 매칭 해결을 위한 알고리즘을 작성하고, 이를 지원하기 위해 CLIENT와 연결되기 위한 포트를 작성한다. 즉, 커넥터의 오퍼레이션명이 된다. 해당 포트를 위한 불변조건, 사전조건, 사후조건을 작성한다[11]. 마지막으로, 해당 커넥터를 테스트하기 위한 테스트 케이스를 작성한다. 적절한 테스트가 되기 위한 입력값과 그 입력값에 의해 예상되는 출력값을 작성한다. 부수적으로, 이는 커넥터 설계자 및 개발자에게 커넥터의 알고리즘 이해에 도움을 준다.

활동 3.2 커넥터의 구조적 모델 작성. 본 활동에서는 활동 3.1에서 작성된 커넥터의 요구사항 명세서를 기반으로 커넥터의 구조적 설계를 한다. 특히, 구조적 부분의 문제인 인터페이스 부분 매칭 문제를 위한 커넥터 설계에 유용하다. 커넥터 설계시, 부분 매칭 문제에 초점을 두고, 해당 부분 매칭 문제가 일어나는 *CLIENT*와 *COMP*의 구조적 모델도 함께 설계한다. 구조적 모델은 6장의 사례연구와 같이 UML의 클래스 다이어그램(Class Diagram)을 사용하여 작성한다[12].

*CLIENT*의 구조적 모델은 커넥터와 관련된 오퍼레이션만을 작성할 수 있다. 식별을 수월하게 하기 위해 «CLIENT» 스테레오 타입 사용이 가능하다. 커넥터 클래스는 포트의 이름을 오퍼레이션으로 표현하고, 커넥터의 기능을 수행하기 위해 필요한 어트리뷰트, 오퍼레이션을 추가한다. 커넥터의 스테레오 타입은 커넥터의 분류자인 인터페이스 어뎁터, 데이타 변환자, 기능 변환자, 워크플로우 핸들러에 따라 «IA», «VR», «FT», «WH» 표현이 가능하다.

활동 3.3 커넥터 동적 모델 작성. 본 활동에서는 커넥터의 메시지 흐름을 설계한다. UML의 시퀀스 다이어그램 (Sequence Diagram)은 객체간의 메시지 패스 관계 및 순서를 보여주기에 적합하다 [12]. 6장 사례연구와 같이 커넥터의 동적 모델 설계는 *CLIENT*와 커넥터 간의 메시지 호출 관계 및 커넥터와 *COMP*간의 메시지 호출 관계를 보여준다. 커넥터에 의해 *CLIENT*의 매개변수들이 어떤 변환 규칙에 의해 부분 매칭 문제가 해결되고, 어떤 경로를 통해 *COMP*에 전달되고, *COMP*의 반환 값들이 어떤 경로를 통해 *CLIENT*에 전달되는지를 명확히 보여준다.

5.4 단계 4. 커넥터 구현

단계 4의 목적은 설계 된 커넥터를 구현하고, 구현 된 커넥터를 컴포넌트와 연결한다. 또한 어플리케이션 요구사항에서 제공하는 기능적, 비기능적 요구사항을 커넥터가 만족시키는지 검증한다.

활동 4.1 커넥터 구현. 이 활동은 단계 3에서 작성된 커넥터의 요구사항 명세서, 커넥터의 설계서를 이용하여 커넥터를 구현한다. 이 활동을 통해서 바이너리 형식의 커넥터가 생산된다. 커넥터의 public 메소드인 포트와 같은 메시지 전달 부분과 부분 매칭 문제 해결을 위한 private 메소드인 변환 규칙 부분을 분리하여 커넥터의 재개발시 유용하게 한다. 컴포넌트 플랫폼이 Enterprise JavaBeans(EJB) 기반[13]이면, 커넥터에서는 *COMP*의 홈 인터페이스와 컴포넌트 인터페이스를 통해 보정된 메시지를 전달하는 로직이 구현된다. 이때 포트, 변환 규칙, *COMP*의 인터페이스를 얻는 부분, Java Naming and Directory Interface(JNDI)에 접속하는 부분과 같이 외부 환경을 위한 메소드로 분류해야 한다.

활동 4.2 커넥터 테스트. 이 활동에서는 작성된 커넥터를 테스트 한다. 첫째, 구현된 커넥터 자체에 대한 테스트를 위해 커넥터 단위 테스트(Unit Test)[14]를 수행한다. 커넥터가 복잡한 경우에는 테스트를 위한 테스트 드라이버(Test Driver)를 작성하여 테스트 한다. 이때 커넥터의 기능성에 초점을 맞춰 테스트 한다. 둘째, 커넥터 단위 테스트를 통해 디버깅 된 커넥터와 컴포넌트를 연결하여 통합 테스트(Integration Test)[14]를 수행한다. *CLIENT*가 원하는 서비스를 제공받고, *COMP*가 서비스를 위해 적절한 메시지 호출이 되는지가 테스트 된다. 이때 커넥터가 기능적 요구사항뿐만 아니라 비기능적 요구사항을 만족하는지 테스트한다.

컴포넌트간에 부분 매칭 문제가 있어 사용 못하는 컴포넌트를 커넥터를 통하여 사용 가능하도록 하였지만, 커넥터로 인해 요구사항의 기능적, 비기능적 요구사항이 만족되지 못한다면, 후보 컴포넌트를 탈락시키거나, 요구사항 명세서를 수정하여 어플리케이션을 개발한다.

6. 사례 연구

본 장에서는 컴포넌트의 부분 매칭 문제를 식별하고, 부분 매칭 문제를 해결하기 위한 사례 연구를 한다. 사례 연구를 위한 도메인은 외환거래(Foreign Exchange), 외화유가증권 등과 같은 국제금융 업무이다. 국제금융은 국내금융과 대조되는 개념으로서 국가와 국가 사이에서 이루어지는 자금의 유통이다. 국제금융 시스템은 그림 10과 같이 외환딜러들이 사용하는 프론트 오피스(Front Office) 시스템과 관련 업무를 수행하고 외부 시스템과 연계하는 백 오피스(Back Office) 시스템으로 구분된다.

국제금융 업무의 경우에는 외국은행과의 업무이기 때

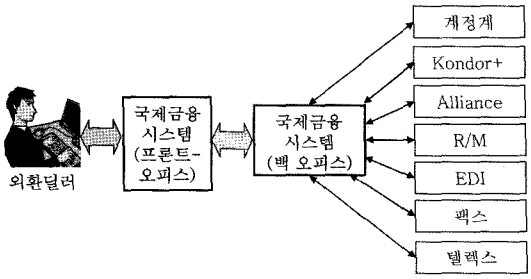


그림 10 국제금융 시스템과 외부 시스템간의 인터페이스

문에 국제적 표준 업무와 인터페이스들이 정의되어 있기 때문에 백 오피스 시스템의 재사용성은 높다. 하지만 프론트 오피스는 각 은행에 적합하게 개발된다. 이러한 프론트 오피스와 백 오피스 사이에서의 부분 매칭 문제를 해결하기 위한 커넥터 사례 연구이다.

6.1 후보 컴포넌트 검색 및 획득

시스템 통합회사에서 S은행을 위한 국제금융 시스템을 개발하려고 한다. S은행은 자사의 프론트 시스템의 변경을 최소화하여 EJB 컴포넌트 기반의 국제금융 시스템을 재개발하려 한다. 외환거래를 위해 S은행에서 요구하는 컴포넌트의 요구 사항은 그림 11과 같다. 외환거래를 위해 필요한 정보는 거래 타입(DealType), 거래 플래그(TradeFlag), 상대방 아이디(CounterPartyID), 상대방 타입(CounterPartyType), 브로커 코드(BrokerCode), 거래방법(DealBy), 거래날짜(DealDate), 딜러아이디(DealerID),

요구 컴포넌트 명세서

1. 컴포넌트 이름 : FXDealSlipMgr
2. 기능성
   - DealSlip 1차 등록 : 외환거래를 위해 딜러가 외환거래 신청서를 작성한다.
   - Deal Slip 2차 등록 : 외환거래를 위해 필요한 추가 2차 정보를 입력한다.
   - Deal Slip 평가 : 외환거래 신청서를 평가한다.
   - Deal Slip 승인 : 외환거래 신청서를 승인한다.
3. 요구되는 인터페이스
   - public boolean insertFirstDealSlip ( ) throws ForexException
     매개변수 : DealType, DealBy, DealDate, BuyAMT, SellAMT, DealerID, . . .
     비고 : DealBy 코드 : (1:브로커, 2:로이터, 3:전화, 4:웹)
   - . . .
4. 관리되는 데이터
   - DealType, DealBy, DealDate, BuyAMT, SellAMT, DealerID, . . .
   - . . .

그림 11 요구되는 국제금융 컴포넌트

구매금액(BuyAMT), 판매금액(SellAMT) 등이 필요하다. 이들 중 본 사례연구에서는 핵심 데이타만 명세한다.

6.2 커넥터의 요구사항 명세서 작성

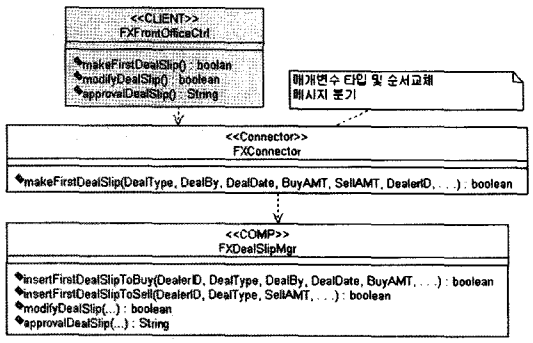
시스템 개발회사에서 획득한 H은행 컴포넌트의 부분 매칭 문제를 식별 한다. 외환거래를 하기 위해 딜러들이 거래 신청서를 작성하는 업무인 'DealSlip을 생성하다.' 유스 케이스를 선택한다. S은행의 프론트 오피스의 요구사항과 H은행에서 획득한 컴포넌트 명세서를 비교하여 부분 매칭 문제를 식별한다. 그리고 식별된 부분 매칭 문제를 해결하기 위한 커넥터의 요구사항 명세서는 표 3과 같다.

표 3 커넥터 요구사항 명세서

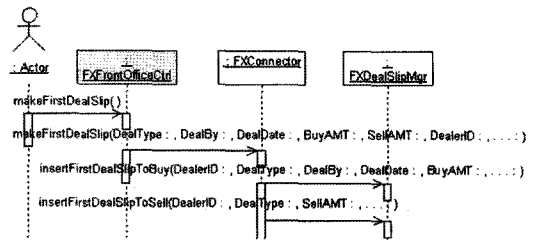
대 항 목	소 항 목	내 용
CLIENT : FXFront-OfficeCtrl	Provided 인터페이스	makeFirstDealSlip( )
	Required 인터페이스	public boolean createFirstDealSlip ( ) throws ForexException 매개변수: DealType, DealBy, DealDate, BuyAMT, SellAMT, DealerID, ...
	비 고	DealBy 코드 : (1:브로커, 2:로이터, 3:전화, 4:웹) DealBy 매개변수는 int형
COMP : FXDealSlipMgr	Provide 인터페이스	public boolean insertFisrtDealSlipToBuy ( ) throws ForexException 매개변수: **DealerID, DealType**, DealBy, DealDate, BuyAMT, ... public boolean insertFisrtDealSlipToSell ( ) throws ForexException 매개변수: **DealerID, DealType**, SellAMT, ...
	비 고	외환 구매와 판매 API가 분리되어 있다. DealBy 코드 : (Broker : 브로커, Reuter:로이터, Phone : 전화, Web : 웹) DealBy 매개변수는 String형
부분 매칭	문 제 점	획득한 *COMP*가 insertFirstDealSlip() 인터페이스를 가지고 있지 않다. DelalerID 매개변수의 순서가 다르다. DealBy의 코드의 의미가 다르다.
Connector: FXConnector	분 류 자	인터페이스 어뎁터(IA), 워크플로우 핸들러(WH)
	해결 방법	(IA) DealerID 매개변수의 위치와 DealBy 매개변수의 형을 변환하여 메시지를 전달한다. (IA) DealBy 코드를 1->Broker, 2->Reuter, 3->Phone, 4->Web으로 변환한다. (WH) 전달받은 매개변수를 구매와 판매 API를 이용하여 각각 호출한다.
	포 트	public boolean makeFirstDealSlip ( ) throws ForexException 매개변수: DealType, DealBy, DealDate, BuyAMT, SellAMT, DealerID, ...

6.3 커넥터의 설계서 작성

표 3과 같이 H은행 컴포넌트의 부분 매칭 문제를 해결하기 위해 작성된 FXConnector의 요구사항 명세서를 이용하여 커넥터를 설계하면 그림 12와 같다. 부분 매칭이 일어나는 FXDealSlipMgr 클래스와 이를 해결하기 위한 커넥터간의 구조적 설계는 (a)와 같다. 또한, 커넥터의 요구사항 명세서에 작성된 커넥터의 이름, 포트 등이 표현된다. 커넥터가 FXFrontOfficeCtrl과 FXDeal-SlipMgr 컴포넌트 사이에서 수행하는 메시지 흐름을 표현한 행위 설계는 (b)와 같다. 이를 통해 커넥터에 전달된 매개변수들이 적절한 메시지 흐름으로 분기 되고, 이 메시지와 함께 해당 매개변수가 컴포넌트로 전달되는 커넥터의 행위를 확인할 수 있다.



(a) 구조적 설계



(b) 행위적 설계

그림 12 커넥터의 구조 및 행위 설계

7. 평 가

소프트웨어 아키텍처에서는 모듈간의 연결을 명세하기 위해 커넥터를 사용하며, 커넥터는 모듈 구현시 실체화 된다[15]. 하지만 구체적인 커넥터의 설계 방법 및 커넥터의 구현에 대한 언급이 미흡하다. 본 논문에서는 컴포넌트간의 부분 매칭을 식별하기 위한 기법을 제안하였으며, 제안된 부분 매칭 식별 기법을 통해 커넥터의 요구사항을 도출하였다. 이를 통해 커넥터 설계하는 프로세스를 제안하였다.

기존의 PORE 기법[4]은 요구사항을 획득하고, 그 요구사항에 만족하는 COTS 컴포넌트를 획득하는 지침을 제공하였다. 하지만, 불일치 현상이 있는 COTS 소프트웨어를 제거하는 방법에 의해 COTS 컴포넌트를 획득한다. 따라서, 부분 매칭 현상이 있지만, 사용 가능한 컴포넌트도 제거 된다. 하지만 본 논문은 표 4와 같이 부분 매칭 현상을 식별 하고, 이러한 문제를 해결하기 위한 방법을 제공하기 때문에 PORE 기법 보다 COTS 컴포넌트의 활용도를 높일 수 있다.

표 4 PORE 기법과의 비교

비 교 항 목	PORE	본 논문
고객 요구사항 식별	지원	지원
후보 COTS 컴포넌트의 만족도 평가	지원	지원
일치되는 컴포넌트 식별	지원	지원
불일치 문제 해결을 위한 방법 식별		지원
불일치 문제가 있지만, 사용 가능한 컴포넌트 식별		지원
불일치 후보 COTS 제거	지원	지원
불일치 컴포넌트를 사용하는 방법	요구사항 변경	커넥터 사용, 요구사항 변경

본 논문에서 제시한 부분 매칭 식별 검증을 위해 컴포넌트 참조 모델에서 발생하는 부분 매칭 뿐만 아니라 다른 커넥터 관련 논문[16,17]을 비교하였다. 표 5와 같이 본 논문에서 제안한 부분 매칭 식별은 속성, 메소드, 인터페이스, 워크플로우의 부분 매칭 뿐만 아니라 다른 논문에서 제안한 커넥터의 종류도 식별될 수 있다.

Spitznager와 Garlan의 커넥터의 유형은 데이타 변환(Data transform), 결합(Splice), 기능 추가(Add a role), 세션화(Sessionize), 집합 변환(Aggregate transform)으로 구성된다[16]. Mehta, Medvidovic, Phadke는 커넥터의 유형은 전달(Communication), 대등(Coordination), 변환(Conversion), 용이성(Facilitation), 절차적 호출(Procedure Call), 이벤트(Event), 데이타 처리(Data Access), 결합(Linkage), 스트림(Stream), 조정자(Arbitrator), 어뎁터(Adaptor), 분배자(Distributor)로 제안하였다[17].

8. 결 론

CBD가 성공하기 위해서는 적절한 컴포넌트가 획득되어 재사용되어야 한다. 불특정 다수의 사용자를 위해 개발된 범용적 컴포넌트는 커스터마이제이션을 기법을 제공하지만, 현실적으로 모든 사용자를 만족시키는 커스터마이제이션을 제공하기는 어렵다. 요구사항과 컴포넌트에서 요구하는 기능성을 커스터마이제이션 기법을 통해 만족시키지 못하는 컴포넌트는 부분 매칭 문제를 가지고 있다.

표 5 다른 논문의 커넥터 식별과의 비교

불일치의 종류	Spitznager	Mehta	제안한 부분 매칭 식별
매개변수의 타입	Data Transfer	Comm., Convers., PC,Event, DA,Stream, Adaptor	I2. 데이타형 불일치
매개변수의 의미	-	-	D1. 매개변수 불일치
반환값의 타입	Data Transfer	Comm., Convers., DA	I4. 반환값의 데이타형 불일치
반환값의 의미	-	-	D2. 반환값의 불일치지원
기능의 불일치	-	-	F1. 기능의 부족, F2. 불필요한 기능 포함, F3. 두가지 혼합(F1, F2)의 경우
커넥터를 이용한 불일치 해결시 발생되는 부작용 발생 유무	-	-	F4. 기능성 제거시 부작용 발생 W5. 워크플로우 변경시 부작용 발생
매개변수의 순서 불일치	Splice	DA, Convers.	I3. 매개변수 순서의 불일치,
오퍼레이션 통합 및 분할	Aggregate	-	W3. 메시지 통합, W4. 메시지 분리
새로운 인터페이스 적용	Add a Role	-	I1. 이름 불일치
워크플로우 추가 및 수정	Aggregate	Coord., PC,Event, Arbitr.	W1. 새로운 워크플로우, W2. 호출 순서
예외처리 기법	-	-	I5. 예외 처리 방법 불일치
영속성의 불일치			P1. 저장 매체의 불일치, P2. 저장 스키마의 불일치 (비고:식별은 되지만, 커넥터에서 미해결)

본 논문에서는 커넥터를 개발하기 위한 프로세스를 제안하였다. 18종류의 부분 매칭의 문제가 발생되는 상황들을 정의하였다. 제안된 분류기법을 이용하여 컴포넌트의 부분 매칭 문제가 식별될 수 있다. 식별된 부분 매칭 문제를 해결하기 위한 커넥터 설계 기법을 제안하였다. 제안된 프로세스를 통해 작성된 커넥터는 컴포넌트의 수정 없이 컴포넌트의 부분 매칭 문제를 해결하여, 기존 PORE 방식을 사용하는 것 보다 더 많은 COTS 컴포넌트의 사용이 가능 할 뿐만 아니라 다른 논문에서 제안한 커넥터의 유형도 식별이 가능하다.

또한 커넥터에 의해 연결된 컴포넌트들은 다른 컴포넌트가 변경이 되어도 커넥터의 수정에 의해 그 영향을 감소 시킬 수 있다. 그리고 이미 연결된 커넥터를 다른 커넥터로 용이하게 교체가 가능하다. 결과적으로 제안된 커넥터 개발 프로세스 및 설계 기법을 통해 효율적으로 COTS 컴포넌트 획득 및 커넥터 개발이 가능하며, 기존에는 사용할 수 없는 COTS 컴포넌트를 획득하고 사용할 수 있게 되었다.

참 고 문 헌

[1] Kim, S., "Lesson Learned from a Nationwide CBD Promotion Project," *Communications of the ACM*, Vol. 45, No. 10, pp. 83-87, 2002.

[2] Kim, S. and Park, J, "A Practical Quality Model for Evaluating COTS Components," *Proceedings of International Association of Science and Technology for Development (IASTED) International Conference on Software Engineering (SE'2003), Innsbruck*, Austria, 2003.

[3] Gokhale, A., et al., "Applying Model-Integrated Computing to Component Middleware and Enterprise Applications," *Communication of ACM, Vol.45, No.10*, 2002.

[4] Heineman, G., and Council, W., *Component-based Software Engineering*, Addison Wesley, 2001.

[5] D'Souza, D. and Wills, A., *Objects, Components, and Frameworks whit UML*, Addison Wesley, 1998.

[6] Gomaa, H., *Desiging Software Product Lines with UML*, Addison-Wesley, pp. 6-7, 2004.

[7] Ohlenbusch, O. and Heineman, G., "Composition and Interfaces within software architecture," *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative research*, November 1998.

[8] Min, H., Choi, S., and Kim, S., "Using Smart Connectors to Resolve Partial Matching Problems

in COTS Component Acquisition," *Proceedings of 7th International Symposium on Component Based Software Engineering(CBSE 2004), LNCS Vol. 3054*, pp. 40-47, 2004.

[9] Kim, S., Her, J., and Chang, S., "A Theoretical Foundation of Variability in Component-based Development," *Information and Software Technology, Vol. 47, pp. 663-673*, July, 2005.

[10] Rosa, N., Justo, G., and Cunha, P., "Incorporating Non-functional Requirements into Software Architectures," *IPDPS 2000 Workshop, LNCS Vol.1800, pp. 1009-1018*, 2000.

[11] Crnkovic, I. and Larsson, M., *Building Reliable Component-Based Software Systems*, Artech House, Inc., 2002.

[12] Rumbaugh, J, et al, *The Unified Modeling Language Reference Manual, Second Edition*, Addison Wesley, 2005.

[13] Sun Microsystems, *Enterprise JavaBeansTM Specification, Ver 2.1*, Sun Microsystems Inc., 2003.

[14] Pressman, R, *Software Engineering,: A Practitioner's Approach*, Six Edition, McGraw-Hill Education, 2005.

[15] Clements, P., et al., *Documenting Software Architectures: Views and Beyond*, Addison Wesley, 2002.

[16] B. Spitznagel and D. Garlan, "A Compositional Approach for Constructing Connectors," *WICSA'01*, 2001.

[17] N. R. Mehta, N. Medvidovic and S. Phadke, "Towards a taxonomy of software connectors," *Proceedings of the 22nd international conference on Software engineering*, June 2000.

민 현 기
정보과학회논문지 : 소프트웨어 및 응용
제 32 권 제 9 호 참조

김 수 동
정보과학회논문지 : 소프트웨어 및 응용
제 32 권 제 9 호 참조