

컴포넌트 조립을 위한 커넥터 자동 생성

Connector Automatic Creation for Component Assembly

채은주, 한정수*, 백순화**

천안대학교 정보기술대학원

천안대학교 정보통신학부*

백석대학교 컴퓨터정보학부**

Chae Eun-Ju, Han Jung-Soo*, Baek Soon-Wha**

Information Technology Graduate School

Cheonan University

Division of Information and Communication,

Cheonan University*

Division of Computer Information,

Baekseok Colleg**

요약

본 논문에서는 컴포넌트의 메소드 호출이나 변경 없이 조립하기 위해 본 논문에서는 컴포넌트를 연결하는 커넥터를 자동생성에 관한 연구를 하였다. 커넥터는 컴포넌트 포트의 정의, 명세와 아키텍처 구조에 의해 자동으로 생성이 된다. 생성된 커넥터를 통해 컴포넌트가 조립이 되는 것이다.

Abstract

In this paper, connector that connect component in paper that serve to assemble without method call or alteration of component offered research of automatic creation. Connector is produced automatically by justice of component port, detail and architecture structure. Through produced connector, component is assembled.

I. 서론

복잡한 응용 프로그램을 빠르게 개발하고 유지 보수 하도록 하기 위해 하나의 소프트웨어 단위를 부품으로 하여 개발하는 컴포넌트 기반 CBD(Component Based Development) 방법이 확산 되고 있다. 또한 이를 위한 컴포넌트 조립에 관한 연구가 진행되고 있다[1]. 컴포넌트 조립은 정적뿐만 아니라 동적 조립에 관해 연구 되고 있다[2]. 컴포넌트 조립은 명세 된 인터페이스 구성을 통해 실행되며 명세 문맥에 의존하여 자동으로 배치 되도록 한다[3].

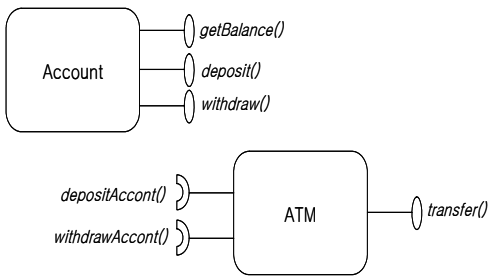
본 논문에서는 컴포넌트 조립 시 컴포넌트의 메소드 호출이나 변경 없이 조립하기 위해 컴포넌트를 연결하는 커넥터에 관하여 연구하였다. 컴포넌트들은 포트를 이용하여 포트의 정의와 명세를 통해 아키텍

처 구조를 생성하여 이를 바탕으로 커넥터를 자동으로 생성된다.

II. ATM 시스템 예제 모델

본 논문에서는 현금 출납기 시스템 모델에 대해 설명한다. Account와 ATM 두개의 컴포넌트로 구성되어 있다. Account 컴포넌트에는 getBalance(), deposit(), withdraw() 3개의 Operation이 있다. ATM 컴포넌트에는 transfer() Operation이 있고 transfer() 에서 필요로 하는 depositAccount()와 withdrawAccount() Operation이 있다. Account 컴포넌트와 ATM 컴포넌트를 연결하기 위해서 커넥터에서는 ATM에 요청으로 들어오는 depositAccount()

와 withdrawAccount() operation을 사용하고 각각의 operation에서는 Account 컴포넌트의 operation인 deposit()과 withdraw() operation을 각각 사용하게 된다. 그림 1은 ATM 시스템 모델을 나타낸다. 여기서 Account 컴포넌트와 ATM 컴포넌트를 연결시킬 커넥터를 생성하기위해 컴포넌트의 정의와 명세, 아키텍처의 구조를 통해 커넥터를 자동 생성할 수 있도록 하였다.



▶▶ 그림 1. ATM 시스템

III. 커넥터 정의와 명세

1. 정의(Definition)와 명세서(Specification)

1) Components and Ports

컴포넌트가 UML class에 명기될 때 스테레오타입 <<component>>으로 표현하여 컴포넌트 명을 표시한다. 컴포넌트는 포트들로 구성된다. 포트(Operation)는 카테고리 기준을 정의되는 연결 부분이다. 포트의 기본형태는 컴포넌트를 provided 또는 required를 operation하도록 정의한 operation port이다. operation port는 스테레오타입 <<operationPort>>로 명세한다. 포트들은 스테레오타입 <<provided>> 또는 스테레오타입 <<required>>에 의해 컴포넌트들을 연결시킨다.

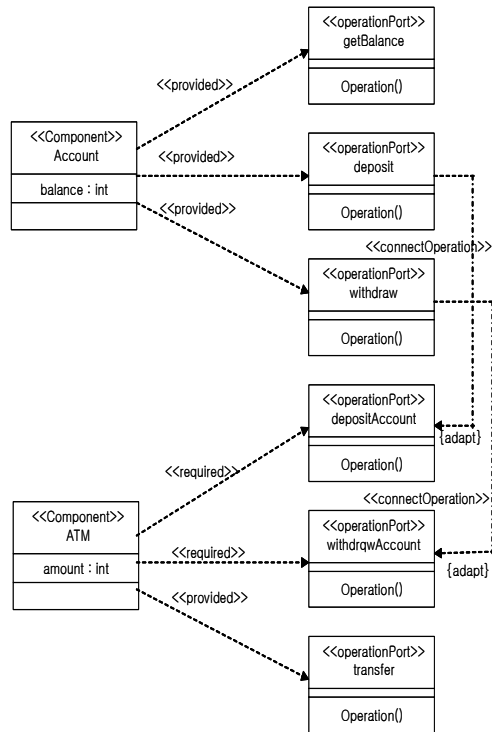
2) Connectors and Assembly

컴포넌트 조립 명세는 컴포넌트 포트들 간의 관계를 정의하는데 있다. required와 provided 포트의 개념은 두 포트의 사이를 연결시키는 것이다. 조립에

중요 요소로서 커넥터를 정의한다. 그것은 또한 호환성 없는 포트들로 적응하도록 사용될 수도 있다. provided 와 required 포트는 UML 의존관계로 스테레오타입 <<connect_type of ports>>으로 커넥터를 명세한다. 각각 컴포넌트를 연결시킬 operation 포트들은 스테레오타입 <<connectOperation>>로 명세한다.

3) Ports Adaptation

커넥터는 두개의 포트를 적응하도록 사용될 수 있다. ATM 모델에서 두개의 포트를 조합을 필요하여 연결할 때 {adapt} 값을 태그(tag)로 표시한다. Adaptation은 맵핑 과정에서 처리된다[3].



▶▶ 그림 2. 컴포넌트 명세, 포트와 커넥터

그림 2를 보면 Account 컴포넌트가 getBalance()와 deposit() 그리고 withdraw()를 provided로 나타

내주고 ATM 컴포넌트는 transfer()를 provided로 나타내주고 depositAccount()와 withdrawAccount()는 required로 나타내주고 있다. Account 컴포넌트의 deposit()과 ATM 컴포넌트의 depositAccount()이 연결되기 위해 스테레오타입 <<connectOperation>>로 나타내고 있다. 이 두 operation을 적용하기 위해 태그(tag)로 {adapt}를 붙여 주었다. 또 Account 컴포넌트의 withdraw()와 ATM 컴포넌트의 withdrawAccount()이 연결되기 위해 스테레오타입 <<connectOperation>>으로 나타내고 있다. 이 두 operation을 적용하기 위해 {adapt}를 붙였다. 그림 2의 예에 대한 구조를 표 1에서 보여주고 있다.

[표 1] 컴포넌트 명세서와 포트와 커넥터

Component	operationPort	connct_type of ports	connectOperation
Account	getBalance	provided	deposit - depositAccount
	deposit	provided	
	withdraw	provided	
ATM	depositAccount	required	withdraw - withdrawAccount
	withdrawAccount	required	
	transfer	provided	

2. 아키텍처 구조

확장된 어떠한 모델에 구조적인 규칙들을 표현한 아키텍처를 추가할 수 있다. OCL (Object Constraint Language) meta-level 제한으로 아키텍처 구조를 명세 한다. 예를 들면, operation 커넥터(의존상태 스테레오타입<<connectOperation>>)는 오직 operation 포트들(클래스 스테레오타입<<operationPort>>)을 결합할 것을 나타내기 위하여, UML 의존관계를 변화할 수 없도록 함으로써 OCL meta-model이 따르도록 추가한다[3]. 그림 3은 아키텍처 구조를 보여준다. 스테레오타입으로 정의된 <<connectOperation>>으로 연결된 각각의 컴포넌트에 대해 supplier인 Account 컴포넌트의 스테레오타입 <<operationPort>>와 client인 ATM 컴포넌트의 스테레오타입

<<operationPort>>로 정의된 operation을 이용한다. supplier 컴포넌트의 객체를 참조하여 client 컴포넌트가 supplier의 객체를 사용한다.

context Dependency

```
inv self.isStereotyped("connectOperation")
implies (( self.supplier → forAll(S:ModelElement
| S.isStereotyped("operationPort")) and
(self.client → forAll(C:ModelElement
| C.isStereotyped("operationPort"))
```

▶▶ 그림 3. ATM 시스템 모델에 대한 OCL

3. 커넥터 자동 생성

커넥터 자동생성은 그림 3의 문맥 의존관계를 통해 그림 4로 표현이 된다. 예를 들어 ATM 모델을 통해 알아보겠다. 우선 {adapt}값이 붙어 있는 스테레오타입을 찾는다. 그리고 그림 3에서 표현한 문맥을 통해 커넥터 자동 생성 규칙 명세서를 작성한다. supplier의 모델요소이거나 스테레오타입 <<operationPort>>로 표현된 operation과 사용자의 모델요소 이거나 스테레오타입 <<operationPort>>로 표현된 operation을 포함한 스테레오타입 <<connectOperation>>의 operation을 이용하여 그림 4와 같이 표현 하였다.

```
deposit - depositAccount {adapt}
supplier : Account Account.deposit
client : ATM ATM.depositAccount
withdraw - withdrawAccount {adapt}
supplier : Account Account.withdraw
client : ATM ATM.withdrawAccount
```

▶▶ 그림 4. 커넥터 자동생성을 위한 규칙 명세서

이러한 명세를 이용해서 커넥터를 생성한다. 생성 과정은 {adapt}라고 표현된 것을 찾는다. 그리고 스

스테레오타입 <<connectOperation>>으로 나타난 operation에서 supplier를 보고 의존하고 있는 컴포넌트를 찾아 커넥터에 그 컴포넌트의 객체를 생성한다. 생성된 객체의 스테레오타입 <<operationPort>>를 찾아 객체 참조를 한다. 그리고 그 객체를 사용할 클래스를 찾아 그 컴포넌트의 스테레오타입 <<operationPort>>를 찾아 의존 상태의 컴포넌트를 사용하게 된다. 예를 들면, ATM 시스템에서 스테레오타입 <<connetOperation>>에 표현된 operation이 Account 컴포넌트의 deposit() 과 withdraw() operation 이면 ATM 컴포넌트의 depositAccount() 과 withdrawAccount() operation 으로 표현된다. supplier의 스테레오타입 <<operationPort>>로 표현되고 있는 Account 컴포넌트의 deposit() 과 withdraw() operation이므로 Account 컴포넌트의 객체를 생성하게 된다. 그리고 client에서 스테레오타입 <<operationPort>>로 표현된 ATM 컴포넌트의 depositAccount()과 withdrawAccount() operation이므로 그 operation 들을 커넥터가 이용할 operation이 된다. 그림 5와 같이 커넥터부분이 생성이 된다.

```
public class connector
{
    Account account = new Account();

    public void depositAccount(int amount)
    {
        account.deposit(amount);
    }
    public void withdrawAccount(int amount)
    {
        account.withdraw(amount);
    }
}
```

▶▶ 그림 5. 커넥터 생성

IV. 결 론

최근 시스템을 재사용하기 위해서 응용 소프트웨어를 컴포넌트 단위로 바꿔 사용하고 있다. 그래서 컴포넌트의 재사용을 하기 위해 컴포넌트 조립에 대한 연구가 활발히 진행되고 있다. 본 논문에서는 컴포넌트 조립을 위해 컴포넌트를 연결 시켜주는 커넥터가 자동으로 생성되게 하였다. 각 컴포넌트를 정의하고 포트를 설정해주며 Operation Port, Connect Port 그리고 adapt를 정의하여 명세서를 작성을 하여 명세 따라 커넥터가 어떻게 자동으로 생성 되는지를 보여 주었다. 커넥터가 자동으로 생성이 되기 때문에 컴포넌트의 메소드 호출이나 변경 없이 컴포넌트 조립이 쉽게 할 수 있다. 자동 생성된 커넥터를 통해 컴포넌트 형상관리 하도록 하는 것이 향후 과제로 남아 있다.

■ 참고문헌 ■

- [1] Edward V. Berard, Essays "Object-Oriented software engineering", Volume 1, Prentice Hall, 1993. "Technique", Information and Software Technology,
- [2] J. Bosch, "Superimposition : A Coponent Adaptaion Vol. 41, Issue 5, March 25, 1999.
- [3] George T. Heineman, "Adaptation and software architecture", Third International Workshop on Software Architecture, Orlando, Florida, November 1998.