레거시 컴포넌트의 유지보수를 위한 AOSD 기반의 Component Connector

김경민° 김태웅 김태공
인제대학교 전산학과
{kmkim, twkim, ktg}@cs.inje.ac.kr

The Component Connector for Maintenance of Legacy Component Based on Aspect Oriented Software Development

Kyung-Min Kim° Tae-Woong Kim Tae-Gong Kim
Dept. of Computer Science, Inje University

요 약

소프트웨어 컴포넌트란 하나 이상의 기능을 갖는 독립적인 소프트웨어이며, 조립을 통해 응용프로그램을 작성할 수 있는 부품 형태의 소프트웨어를 말한다. 이미 개발되어진 컴포넌트에 기능을 추가하거나 확장하기 위해서는 새로운 컴포넌트를 개발하여 기존의 컴포넌트와의 조립을 통하여 이를 실현하거나 이미 개발되어진 컴포넌트를 수정해야 한다. 더구나 추가하고자 하는 기능이 컴포넌트의 내부에 흩어져 있는 경우에는 컴포넌트를 수정해야하는 단점이 있다. 이에 본 논문에서는 Aspect를 이용하여 컴포넌트의 기능 추가 및 확장을 가능하게 하는 컴포넌트들 간의 조립을 위한 Component Connector를 제안한다. 컴포넌트들 간의 조립정보를 표현하는 Component Connector 모델을 제시하고 요청된 조립 서비스를 수행하는 Component Connector의 실현 클래스를 정의한다. 이것은 컴포넌트 플랫폼 독립적인 모델로 나타내어지며 적용사례를 통하여 다양한 형태로의 플랫폼 종속적인 모델로 변환 가능함을 검증하고자 한다.

1. 서 론

CBSD(Component Based Software Development) 방법론은 표준화되고 검증된 컴포넌트를 활용하여 소프트웨어 개발 생산성을 향상시키기 위한 방법의 하나로 소프트웨어를 부품화하고 이를 조립 합성하여 새로운 어플리케이션을 개발한다[1]. 즉, 경험이 많은 개발자에 의해 만들어진 컴포넌트들을 재사용함으로써 신뢰성, 안정성 및 유지보수성이 매우 뛰어난 소프트웨어를 개발할 수 있는 것이다.

하지만 레거시 컴포넌트에 스케줄링, 트랜잭션, 보안, 로깅, 문맥의존형 에러처리와 같이 여러 다른 모듈들에 걸쳐져서(scattering) 구현되어야 하는 기능들을 확장하기 위해서는 기존의 컴포넌트 조립방법으로는 불가능하며 연관된 모든 컴포넌트를 분석하고 레거시 컴포넌트를 수정하여 새롭게 구현해야 하는 문제가 생긴다.

이에 본 논문에서는 레거시 컴포넌트의 유지보수를 위해 컴포넌트 조립에 AOSD(Aspect Oriented Software Development) 개념을 적용한다. AOSD는 소프트웨어를 개발하는데 있어서 여러 다른 모듈들에 걸쳐져서 구현되어야 하는 기능 단위인 Crosscutting Concern을 위해 필요한 부분만을 임의로 잘라서 이를 바탕으로 모듈화 하여 시스템을 개발 가능하게 해주는 새로운 개념의 기술이다[2,3].

본 논문에서는 컴포넌트에서 기능을 위한 부분과 다른 컴포넌트와의 연결을 위한 부분을 분리하여 컴포넌트들 간의 조립정보를 표현하는 Component Connector를 제안한다. Component Connector의 정의를 위해 AOSD 기반의 컴포넌트 조립을 위한 정보를 분석하고 이에 대한 메타 모델을 정의한다. 이를 바탕으로 UML을 확장하여 Component Connector 모델을 제시한다. 그리고 Component Connector 모델을 기반으로 요청된 조립 서비스를 수행하는 Component Connector의 실현 클래스를 정의하고, 실제 조립 시 사용되는 컴포넌트의 종류에 따라 다양한 플랫폼으로 산출해본다.

2. 컴포넌트의 유지보수를 위한 Component Connector

본 논문에서는 레거시 컴포넌트의 유지보수를 위해 기능을 위한 부분과 다른 컴포넌트와의 연결을 위한 부분을 분리하여 컴포넌트들 간의 조립정보를 표현하는 AOSD 기반의 Component Connector를 제안한다. AOSD 개념에 따라 레거시 컴포넌트는 Core 컴포넌트에 해당되며 유지보수 시 여러 다른 모듈들에 걸쳐져서 구현되어야 하는 기능은 Aspect 컴포넌트에 해당된다.

*Component Connector*는 레거시 컴포넌트에 새로운 기능을 추가하기 위한 것으로써 그림 1과 같이 Provide와 Provide 인터페이스 사이의 상호작용을 정의한다. 이것은 AOSD에서 정의하는 결합에 관한 방법을 활용한 것으로 의존관계 없이 독립적으로 구현된 컴포넌트들 간에도 조립에 의한 개발이 가능하여 레거시 컴포넌트의 기능 확장에 효율적이다.

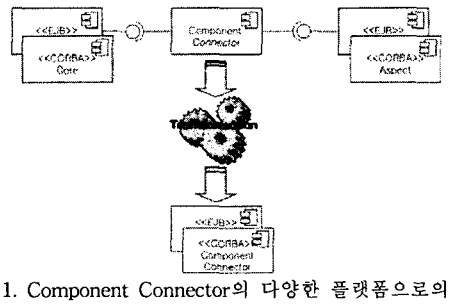


그림 1. Component Connector의 다양한 플랫폼으로의 변환

그리고 요청된 조립 서비스를 수행하는 Component Connector는 조립 시 사용되는 컴포넌트의 종류에 따라 다양한 플랫폼으로의 산출이 가능하다. 이것은 플랫폼 독립적인 Component Connector 모델을 제시함으로써 EJB, CORBA 등 다양한 종류의 컴포넌트에 적합한 플랫폼 종속적인 *Component* Connector를 산출 가능하게 하여 컴포넌트의 조립에 의한 개발을 더욱 효과적으로 지원한다.

2.1 AOSD 기반의 컴포넌트 조립을 위한 정보

컴포넌트를 실제 조립하기 위해서는 조립하고자 하는 두 컴포넌트, Core 컴포넌트와 Aspect 컴포넌트의 인터페이스에 대한 정보가 필요하다. 그리고 조립하고자 하는 컴포넌트의 기능이 Core 컴포넌트의 구조상에서 어떠한 위치, 시간, 상황에 의존하여 최종적으로 원하는 행위로 변형시킬 것인가에 대한 정보가 필요하다.

위치 정보는 Aspect 컴포넌트가 Core 컴포넌트에 결합되는 위치로서 Core 인터페이스의 오퍼레이션 호출 정보를 의미한다. 시간 정보는 결합되는 시점에 관한 정보로서 Core가 실행되기 전에 Aspect를 실행하는 'before', Core가 실행된 후에 Aspect를 실행하는 'after'를 나타낸다. 상황 정보는 인터페이스의 오퍼레이션이 호출된 경우를 의미하는 'call', 인터페이스의 속성 값이 읽혔을 경우를 의미하는 'get', 속성 값이 변경되었을 경우를 의미하는 'set'을 나타낸다.

이렇게 분석된 조립 정보를 기반으로 정의된 Component Connector의 메타 모델은 그림 2와 같다.

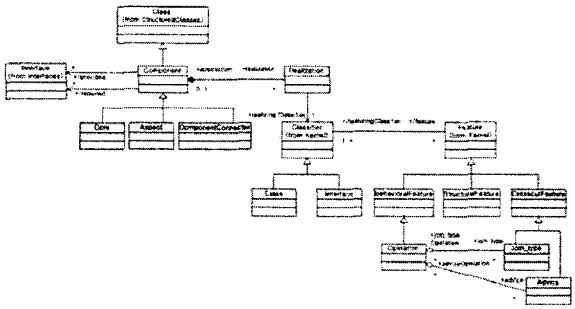


그림 2. Component Connector의 메타 모델

그림 2에서 'Join_type'이 존재하는 경우는 반드시 'advice' 정보가 필요하다. 이에 대한 OCL[4] 제약조건은 다음과 같다.

```
context Operation inv:
(self.join_type->exists('call')) implies
self.advice->notEmpty
```

그리고 'Advice'가 'before'인 경우는 'Operation' 이름에 'Iaspect'가 포함된 인터페이스의 오퍼레이션을 먼저 실행하고, 'after'인 경우는 'Operation' 이름에 'Icore'가 포함된 오퍼레이션을 먼저 실행해야 한다. 이에 대한 제약 조건은 다음과 같다.

```
context Operation inv:
(self.advice->exists('before')) implies
Advice.adviceOperation.name=Iaspect

context Operation inv:
(self.advice->exists('after')) implies
Advice.adviceOperation.name=Icore
```

2.2 Component Connector

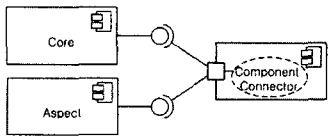
2.1에서 분석된 컴포넌트 조립 정보들을 UML2.0[5]의 확장 메커니즘과 컬레보레이션 템플릿을 이용하여 정의한다. 컬레보레이션 템플릿은 요소들이 협력을 위해 어떻게 상호작용하는가에 대해 명세 하는 것으로 본 연구에서는 내부적으로 시퀀스 다이어그램을 이용하여 정의한다.

표 1은 조립 정보에 대한 UML 확장을 나타낸다.

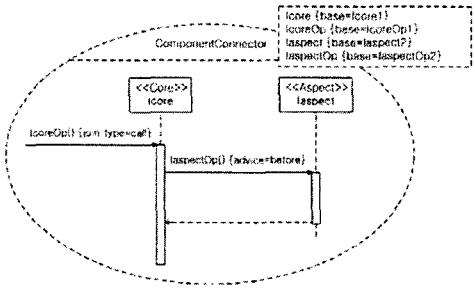
표 1. 조립 정보에 대한 UML 확장

조립정보	UML 확장
컴포넌트정보	<<Core>>, <<Aspect>>
인터페이스정보	<<Icore>>, <<Iaspect>>, Icore {base='Icore1'}
위치정보	IcoreOp {base='IcoreOp1'}
시간정보	{advice='before or after'}
상황정보	{join_type='call or get or set'}

그림 3은 본 논문에서 제안하는 Component Connector 모델이다. (a)처럼 의존관계가 전혀 없는 Core와 Aspect 컴포넌트는 Component Connector를 통해 조립될 수 있다. 포트를 통해 연결된 두 인터페이스는 (b)처럼 Component Connector의 컬레보레이션 템플릿으로 상호작용이 명세 된다.



(a) 포트로 연결된 두 컴포넌트의 인터페이스



(b) 'Component Connector' 컬레보레이션 템플릿에서의 상호작용 명세

그림 3. Component Connector 모델

그림 3의 (b)는 실제 조립하고자 하는 두 컴포넌트가 구조상에서 어떠한 위치, 시간, 상황에 의존하여 결합되는지 나타내고 있다. Core 컴포넌트의 해당 인터페이스 'Icore'의 오퍼레이션 'IcoreOp()'가 호출('join_type=call') 될 때 마다 Core가 실행되기 전에('advice=before') Aspcet 컴포넌트의 인터페이스 'Iaspect'의 오퍼레이션 'IaspectOp()'이 수행되는 것을 의미한다.

이렇게 명세된 조립 정보를 기반으로 요청된 조립 서비스를 수행하기 위해서는 클래스에 의해 실현되어야 한다. 그림 4는 Component Connector의 내부 실현 클래스다. 포트를 통해 요청된 조립 서비스는 'ComponentConnector' 클래스에게 위임되어 실현된다.

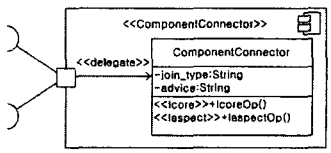


그림 4. Component Connector의 내부 실현 클래스

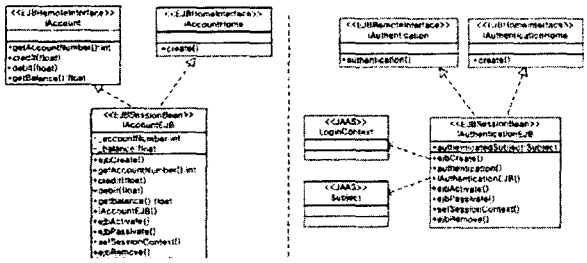
'ComponentConnector' 클래스는 조립하고자 하는 Core와 Aspect 컴포넌트의 인터페이스 오퍼레이션 정보를 모두 가지게 되며 조립 시 필요한 'join_type'과 'advice' 정보를 속성으로 가진다. 이 Component Connector는 조립 시 사용되는 컴포넌트의 종류에 따라 다양한 플랫폼으로 산출될 것이다.

컴포넌트의 기능과 조립에 관한 정보를 분리함으로써 복잡함을 감소시키고, 한 가지 Component Connector 모델에 대하여 다양한 구현을 제공함으로써 컴포넌트의 변경 없이 필요에 따라 적절한 Component Connector 구현을 이용할 수 있어 컴포넌트의 재사용성이 높아진다.

3. 적용 사례

본 논문에서 제시한 Component Connector 모델을 간단한 은행 시스템에 적용해본다. 입금과 출금을 수행하는 기존의 은행 시스템인 Core 컴포넌트에 인증에 관한 기능인 Aspect 컴포넌트를 추가 하고자 한다. EJB 기반의 컴포넌트에 적용해봄으로써 해당 플랫폼에 대한 Component Connector가 산출되는 것을 확인해 본다.

그림 5는 EJB 기반의 간단한 은행 시스템에서의 Core 컴포넌트와 Aspect 컴포넌트를 나타내고 있다.



(a) 은행시스템에 대한 Core컴포넌트 (b) 인증에 대한 Aspect컴포넌트

그림 5. EJB 기반의 은행 시스템

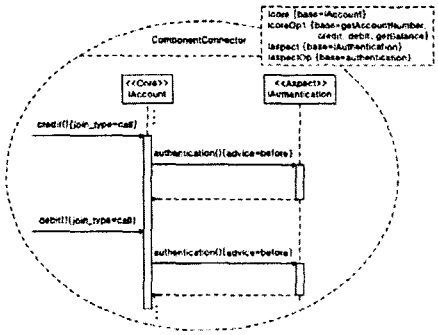


그림 6. 조립 정보를 나타내는 Component Connector 모델

그림 6은 그림 5에 대한 조립 정보를 나타내는 Component Connector 모델로써 Core와 Aspect 컴포넌트 간의 결합 시에 필요한 위치, 상황, 시간에 관한 정보를 나타내고 있다. Core의 IAccount 인터페이스에 있는 모든 오퍼레이션(getAccountNumber(), credit(), debit(), getBalance())이 호출({join_type=call})될 때 마다 Aspect의 IAuthentication 인터페이스의 authentication() 오퍼레이션이 먼저({advice=before}) 수행되어 인증과정을 거치게 된다.

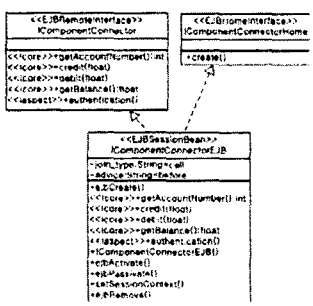


그림 7. Component Connector의 EJB 기반 내부 실현 클래스

그림 7은 그림 6에 명세된 조립 정보를 기반으로 요청된 조립 서비스를 수행하기 위한 Component Connector의 내부 실현 클래스다. 사용된 컴포넌트 종류에 따라 EJB기반의 Component Connector로 산출된다.

컴포넌트의 기능과 조립에 관한 정보를 분리함으로써 복잡함을 감소시키고, 한 가지 Component Connector 모델에 대하여 다양한 실현 클래스를 제공함으로써 컴포넌트의 조립에 의한 개발을 더욱 효과적으로 지원한다.

4. 결 론

본 논문에서는 레거시 컴포넌트의 유지보수를 위해 컴포넌트에서 기능을 위한 부분과 다른 컴포넌트와의 연결을 위한 부분을 분리하여 컴포넌트들 간의 조립정보를 표현하는 AOSD 기반의 Component Connector를 제안했다. AOSD 기반의 컴포넌트 조립을 위한 정보를 분석하고 이에 대한 메타 모델을 정의했다. 이를 바탕으로 UML을 확장하여 Component Connector 모델을 제시하고 요청된 조립 서비스를 수행하는 Component Connector의 실현 클래스를 정의했다. 또한 다양한 플랫폼 종속적인 모델로의 변환 가능함을 검증하고자, 적용 사례를 통하여 EJB 플랫폼으로 산출해 보았다.

Component Connector 모델은 레거시 컴포넌트에 새로운 기능을 추가하기 위한 조립 정보를 표현한 것으로써 Provide와 Provide 인터페이스 사이의 상호작용을 정의한다. 이것은 AOSD에서 정의하는 결합에 관한 방법을 활용한 것으로 의존관계 없이 독립적으로 구현된 컴포넌트들 간에도 조립에 의한 개발이 가능하여 레거시 컴포넌트의 기능 확장에 효율적이다.

요청된 조립 서비스를 수행하는 Component Connector는 조립 시 사용되는 컴포넌트의 종류에 따라 다양한 플랫폼으로의 산출이 가능하다. 이것은 플랫폼 독립적인 Component Connector 모델을 제시함으로써 EJB, CORBA 등 다양한 종류의 컴포넌트에 적합한 플랫폼 종속적인 Component Connector의 실현 클래스를 산출 가능하게 하여 컴포넌트의 조립에 의한 개발을 더욱 효과적으로 지원한다.

본 논문에서 제안하는 Component Connector는 레거시 컴포넌트에서 여러 모듈들에 걸쳐져서 수행되는 기능을 추가해야 되는 경우 컴포넌트의 수정 없이 기능 확장을 가능하게 해준다. 그리고 컴포넌트 간의 조립 정보를 분리함으로써 실제 컴포넌트의 재사용을 높일 수 있다. 컴포넌트 간의 결합력과 상호 의존도가 약해져 적응성이 향상되어, 이를 이용하면 다양한 컴포넌트를 서로 손쉽게 연결할 수 있다. 즉, 레거시 컴포넌트의 유지보수 시 컴포넌트의 조립을 효과적으로 지원해줌으로써 기술의 변화에 효율적으로 대응할 수 있으며 관리 및 유지 보수가 쉽게 된다.

앞으로 Component Connector를 이용한 레거시 컴포넌트와 Aspect 컴포넌트와의 동적 결합을 가능하게 하는 미들웨어에 대한 연구가 필요하겠다.

참고문헌

[1] Booch G., Kozaczynski Wojtek, "Component-Based Software Engineering", IEEE Software, pp.34-36, 1998

[2] AOSD conference, http://www.aosd.net

[3] 이준상, "미래 소프트웨어 개발기술 : Aspect-Oriented Programming과 Subject-Oriented Programming", 정보처리학회지 제10권 제5호, 2003. 9

[4] Object Management Group, "OCL 2.0 Specification", OMG document ptc/2005-06-06, 2005

[5] Object Management Group, "UML 2.0 Infrastructure, UML 2.0 Superstructure Specification", OMG document ptc/03-09-15, 2003