DOI: 10.3745/KIPSTD.2008.15-D.5.659

트리 컴포넌트 모델 : 하이브리드 메시지 전달을 사용한 컴포넌트 조합

허 제 민[†] · 김 지 홍[††]

요 약

최근 컴포넌트 간의 약 결합을 지원하는 Exogenous 커넥터 기반의 컴포넌트 모델이 제안되었다. 이 모델은 커넥터에서 메소드 호출의 시작과 결과를 관리하여 제어와 계산을 분리한다. 하지만 컴포넌트 사이의 연결 계층과 커넥터 수가 증가 할수록 시스템을 구성하는 객체의 수가 크게 증가하는 문제점을 가지고 있다. 본 논문에서는, 직접 메시지 전달과 간접 메시지 전달이 혼합된 하이브리드 메시지 전달을 사용한 트리 컴포넌트 모델을 제안한다. 이는 컴포넌트를 인터페이스들로 랩핑하여 모든 제어가 계산과 분리되어 인터페이스 레퍼런스를 통해서만 이루어진다. 이렇게 연결된 컴포넌트 사이의 조합은 항상 트리 구조를 이루는 특징이 있다. 아울러 구현 예와 비교 평가를 통해 트리 컴포넌트 모델이 실용적으로 적용 가능하고 시스템의 구성과 메시지 전달을 중재하는 객체 수의 감소를 확인할 수 있었다.

키워드 : 소프트웨어 공학, CBD, 컴포넌트 모델, 컴포넌트 조합, 소프트웨어 컴포넌트

Tree Component Model : Component Composition with Hybrid Message Passing

Huh, Je-Min[†] · Kim, Ji-Hong[††]

ABSTRACT

Recently, the component model based on the Exogenous Connector has been proposed in which controls are separated from computation by managing the beginning and result of method calls in the connector. Although it could be loosely coupled between components, it has a problem that is a potential preponderance of element objects of the system by increasing the number of connectors and connection levels. In this paper we propose the Tree Component Model with the Hybrid Message Passing that combines direct and indirect message passing. In our model, components are wrapped by interfaces and controls are separated from computation by only using their interface references. There is a unique feature that the composition structure of components becomes the tree always. As a result of demonstration and comparison, it is found that the Tree Component Model is applicable practically and decreases objects to mediate message passing and build the system.

Keywords : Software Engineering, CBD, Component Model, Component Composition, Software Component

1. 서 론

현재 컴포넌트 모델에 관한 많은 연구가 진행되어 상당한 성과를 이루었다 [1, 2]. 그러나 기존의 모델들은 독점적인 프레임워크와 컴포넌트 명세서를 가지고 있어 이를 사용하는데 많은 학습의 시간이 필요하다. 게다가 이들의 컴포넌트는 재사용하기 쉽지 않거나 조합 메커니즘이 잘 정의되지 않아 시스템 적으로 적용하기가 어렵다[3, 4].

특히 EJB, CORBA, COM, KobrA 등의 모델들은 RPC 메소드와 이벤트 위임과 같은 직접 메시지 전달 방법을 사용하여 컴포넌트들을 연결하였기 때문에 이는 순환 호출이 발생하고 제어와 계산이 분리되지 않는 등의 심각한 문제를 가지고 있어 컴포넌트의 재사용을 어렵게 하였다[5, 6]. 이외 JavaBeans, ADLs, Koala, PIN, PECOS 등의 모델들은 커넥터 또는 추상화의 다른 단계에서 커넥터로 해석될 수 있는 조합 연산자를 통해 컴포넌트 사이를 연결하는 간접 메시지 전달 방법을 사용하였다. 이들은 커넥터에서 컴포넌트의 사이의 상호작용 또는 의사소통을 은닉하고 컴포넌트에서 계산을 은닉하지만 다른 컴포넌트의 메소드를 호출할 경우에 발생하는 제어의 시작이 컴포넌트에서 이루어지기 때문에 여전히 제어의 관점에서 컴포넌트 사이의 강한 결합이 발생하는 문제를 가지고 있다 [5, 6].

기존의 커넥터를 사용한 모델들에서 제어에 의한 컴포넌트

† 준 회 원 : 경원대학교 전자계산학과 석사
†† 정 회 원 : 경원대학교 IT대학 소프트웨어학부 교수(교신저자)
논문접수 : 2008년 8월 5일
수 정 일 : 1차 2008년 9월 18일
심사완료 : 2008년 9월 19일

사이의 강한 결합을 가지는 문제를 해결하기 위해 Exogenous 커넥터를 사용한 컴포넌트 모델(이하 Ex 모델)이 제안되었다[5, 6, 7, 8]. 이 모델은 다른 컴포넌트의 메소드 호출과 같은 제어의 시작을 포함한 모든 제어가 커넥터에서 이루어지고 서비스 로직인 계산은 모두 컴포넌트에서 이루어진다. 즉, 제어와 계산이 완벽히 분리된다. 하지만 계층의 단계가 증가 할수록 커넥터의 수가 증가하여 단계의 차이가 커짐에 따라 시스템 구성과 메시지 전달을 중재하는 객체가 크게 늘어나는 문제를 가지고 있다.

본 논문에서는 Ex 모델과 같이 제어와 계산을 분리하여 컴포넌트 사이의 약 결합이 가능하고 동시에 연결의 수가 증가하여도 커넥터의 수를 억제하여 시스템 구성과 메시지 전달시의 중재 객체의 수를 줄이는 새로운 컴포넌트 모델을 제안한다. 이를 위해 직접 메시지 전달과 간접 메시지 전달이 혼합된 하이브리드 메시지 전달을 사용한다. 또한 컴포넌트 사이를 연결한 조합 구조가 트리[9]를 이루도록 하여 직접 메시지 전달의 문제점인 순환적 호출 또한 방지한다. 결과적으로 컴포넌트 사이의 연결을 통해 조합된 구조는 항상 트리를 이루는 특징을 가지는 이 모델을 트리 컴포넌트 모델이라 하고 이를 위한 컴포넌트를 트리 컴포넌트라고 부른다.

본 논문의 구성은 2장에서 관련 연구로 소프트웨어 컴포넌트 모델의 정의와 메시지 전달 방법에 관해 알아본다, 3장에서는 하이브리드 메시지 전달을 사용한 트리 컴포넌트 모델을 제안한다. 4장에서는 트리 컴포넌트 모델의 간단한 구현의 예를 들고 5장에서는 Ex 모델과의 비교를 통해 트리 컴포넌트 모델의 평가를 제시한다, 마지막으로 6장에서 논의 및 결론을 기술한다.

2. 관련 연구

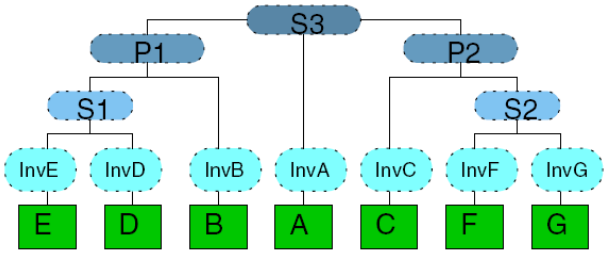
소프트웨어 컴포넌트는 컴포넌트의 특성을 소프트웨어에 적용한 것으로 구체적 특성은 다음과 같다 [1].

- 독립적 배포의 단위
- 3-자 조합 단위
- 외부에서 관찰 가능한 상태가 없다

위의 모든 특성을 포함한 현재 가장 널리 받아들여지고 있는 소프트웨어 컴포넌트의 정의는 다음과 같다[3]. "소프트트웨어 컴포넌트는 명확한 환경 의존성과 합의에 의해 명세화 된 인터페이스를 사용한 단지 조합의 단위이다. 소프트웨어 컴포넌트는 독립적으로 배포될 수 있고 3-자에 의해 조합되는 대상이다[1]." 이 정의를 바탕으로 소프트웨어 컴포넌트는 이해되어 왔다. 하지만 실제로 적용하기 위해서는 구체적인 모델이 필요하다. 따라서 본 논문의 바탕이 되는 소프트웨어 컴포넌트 모델과 현재 사용되는 메시지 전달 방법인 직접 메시지 전달과 간접 메시지 전달을 설명할 것이다.

2.1 소프트웨어 컴포넌트 모델

모델은 실재 세상의 단순화된 표현이다[10]. 현재의 모델들은 크게 객체지향 프로그래밍에서와 같이 컴포넌트들이



(그림 1) Exogenous Connection

객체인 모델들 또는 소프트웨어 아키텍처들과 같이 컴포넌트들이 아키텍처 단위인 모델들 두 가지 범주로 나뉜다. 이들 소프트웨어 컴포넌트 모델은 컴포넌트는 정해진 규칙과 지침에 따라 상호 작동할 수 있게 한다[11].

2.1.1 Exogenous 커넥터를 사용한 컴포넌트 모델

기존에 커넥터를 사용한 모델의 문제점을 해결하기 위해 Exogenous 커넥터를 사용한 모델은 다음의 특징을 가진다[5].

- 제어와 처리가 혼합되지 않는다.
- 외부로부터 메소드 호출이 시작된다.

위의 특성을 통해 제어는 커넥터에서만 발생하고 계산은 컴포넌트에서만 발생한다. 그리고 커넥터를 중심으로 컴포넌트들이 계층적으로 연결되고 모든 컴포넌트들은 커넥터를 통해서 제어가 이루어진다. 커넥터의 종류로는 Inv(호출), S(셀랙터), P(파이프) 커넥터가 존재하고 컴포넌트와 커넥터 사이의 조합 계층 구조는 (그림 1)[6, 7]과 같다.

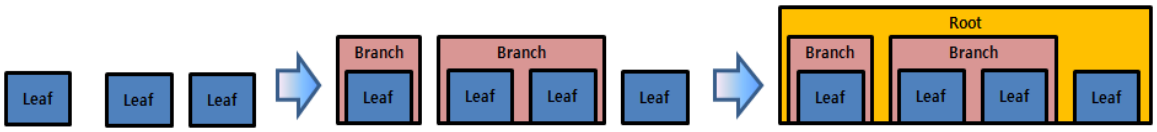
Exogenous 커넥터를 사용한 계층적 구조는 본 논문에 큰 동기가 되었다. 그러나 이 모델은 계층이 증가할수록 커넥터의 잠재적인 큰 증가를 피할 수 없는 문제가 있다.

2.2 직접 메시지 전달과 간접 메시지 전달

현재 컴포넌트 모델에서 연결을 위한 방법은 두 가지 분류로 나뉜다 : 직접 메시지 전달, 간접 메시지 전달 [5].

직접 메시지 전달을 사용하면 컴포넌트 사이의 의사소통시 자신과 관련된 컴포넌트를 직접호출 하여 전달하므로 메시지 전달 방법이 매우 효율적이다. 하지만 컴포넌트들이 서로 의존하여 각각의 컴포넌트를 재사용하기 어려운 문제점이 있고 A 컴포넌트의 메소드 x가 B 컴포넌트의 메소드 y를 호출하고 y는 C 컴포넌트의 메소드 z를 호출면 z는 A의 x를 호출(즉, x→y→z→x)하는 등의 순환적 호출 같은 문제를 방지하는 방법이 없어 개발자의 능력에 의존할 뿐만 아니라 컴포넌트들 내부에 제어와 계산이 혼합되어 재사용을 어렵게 한다.

간접 메시지 전달은 컴포넌트가 커넥터에 의존하기 때문에 커넥터에서 컴포넌트 사이의 상호작용 또는 의사소통을 은닉하고 컴포넌트에서 계산을 처리한다. 이는 컴포넌트 사이의 의존성을 제거할 수 있어 컴포넌트를 각각 재사용 가능하게 한다. 하지만 이는 커넥터라는 중간의 객체를 생성하여 연결하므로 객체의 추가적인 생성과 같은 오버헤드가



(그림 2) 트리 컴포넌트 모델의 확장 과정

발생하고 컴포넌트의 연결이 증가하면 커넥터들의 수 또한 증가하므로 커넥터들을 관리하는 추가적인 방법 또한 필요하게 된다. 또한 현재 Ex 모델 외에 간접 메시지 전달을 사용하는 모든 모델들은 컴포넌트 내부에서 메소드 호출이 시작되어 제어의 관점에서 여전히 강한 결합을 가진다.

3. 트리 컴포넌트 모델

CBD의 목적은 이미 구성된 작은 부품들을 조합하여 복잡한 시스템을 이루기 위한 것이다[1, 3, 4, 12, 13]. 마치 레고 블록 쌓아서 문과, 지붕과 같은 특별한 기능을 가지는 블록의 단위를 만들고 이 또한 조합되어 레고 블록으로 만든 집이 되는 것과 같이 작은 컴포넌트들이 조합되어 좀 더 복잡한 컴포넌트가 되고 이들 또한 조합되어 최종적으로 소프트웨어 시스템이 되는 것이다. 이를 가능하게 하기위해 레고 블록들을 쉽게 연결할 수 있는 쉬운 방법이 제공 되는 것과 같이 소프트웨어 컴포넌트 모델에서 컴포넌트의 재사용을 위해 연결하기 쉬운 조합 메커니즘을 정의하는 것이 필요하다. 또한 컴포넌트 사이의 조합이 추적가능하고 예측 가능한 구조를 이루도록 한다면 시스템의 구조와 컴포넌트 사이의 관계를 쉽게 파악할 수 있고 나아가 시스템적인 컴포넌트 조합도 가능할 것이다.

하위 절을 통해 트리 컴포넌트 모델의 정의에 관해 자세히 알아보고 트리 구조를 이루기 위한 하이브리드 메시지 전달의 구체적인 메커니즘에 관해 설명할 것이다. 그리고 이를 기반으로 한 구체적인 구현 방법을 제시할 것이다.

3.1 트리 컴포넌트 모델의 정의

트리 컴포넌트 모델은 컴포넌트 사이의 조합이 반복되어도 언제나 트리 구조로 연결되는 특징을 가지는 모델이다. 이는 간단한 연결 방법의 반복을 통해 복잡한 구조를 가지는 익숙한 계층 구조인 트리의 특성을 상속하며 다음의 규칙을 따른다.

1. 서비스 인터페이스의 타입으로만 컴포넌트를 생성한다.
2. 트리 컴포넌트 모델의 인터페이스 레퍼런스를 통해서 모든 제어를 수행한다.
3. 트리 컴포넌트 모델의 인터페이스들로는 형 변환을 할 수 없다.
4. 컴포넌트들은 독립적으로 사용가능하도록 개발된다.

이들 규칙은 트리 구조를 이루고 유지하는 규칙이다. 특히 규칙 4를 지키기 위해서 컴포넌트가 생성될 때 서비스를 제공하기 위해 필요한 모든 정보(기본 속성 값, 데이터, 필요한 객체)를 생성자의 인자로 제공받아 생성된다. 이들은 생성자의 인자들만 제공된다면 독립적으로 생성가능하고 다른 컴포넌트와 아무런 관련이 없이도 사용할 수 있도록 구현된다. 그리고 이렇게 생성된 컴포넌트 객체는 반드시 하나의 상위 컴포넌트를 위해서만 사용된다. 트리 컴포넌트 모델이 기존의 트리 구조와 다른 점은 루트에서 시작되어 아래로 트리를 확장하는 것이 아니라 (그림 2)와 같이 반대로 아래에서부터 위로 트리를 확장해 나가는 것이다. 이는 트리 컴포넌트 모델에서 새로 만들어질 컴포넌트는 저장소에 클래스형태로 저장되어 있는 이미 존재하는 컴포넌트를 내부에 객체로서 포함하는 랩핑 방법을 사용하기 때문이다. 따라서 트리 컴포넌트 모델에서 트리를 이루기 위한 세부 방법과 필요한 인터페이스를 식별하고 연결 구조를 설명할 것이다.

3.1.1 트리를 위한 방법과 인터페이스

트리 구조를 이루기 위한 방법으로는 컴포지트 패턴[14, 15]이 있다. 이는 객체들을 트리 구조로 구성하여 부분과 전체를 나타내는 계층구조로 만드는 방법이다[15]. 그러나 컴포지트 패턴을 컴포넌트 조합에 적용하여 트리 구조를 이루는 것은 가능하지만 이는 상위에서 하위로의 메시지 전달만 가능 할 뿐이다. 컴포넌트는 서로 커뮤니케이션이 가능해야 하기 때문에 하위에서 상위로 메시지 전달을 하기 위한 방법이 필요하다.

트리 구조를 컴포넌트에 적용하기 위해 트리를 연결 관점과 구성하는 노드의 관점에서 살펴보면 트리 구조는 위에서 아래로의 연결과 아래에서 위로의 연결이 존재한다. 그리고 트리를 구성하는 노드들은 3가지 종류가 있다. 이들은 루트, 잎, 그리고 중간의 노드들이다. 루트는 위에서 아래로의 연결만 필요하고, 잎은 아래에서 위로의 연결만 필요하다. 그리고 중간의 노드는 둘 다 필요할 것이다. 그러므로 제어를 위한 연결 인터페이스는 상위에서 하위로 또는 하위에서 상위로의 연결만 필요하다. 그리고 상위에서 하위로의 연결에는 트리 구조를 위한 연결 인터페이스뿐만 아니라 상위에서 하위의 서비스 로직에 접근할 수 있는 서비스 인터페이스 또한 제공해야한다. 또한 컴포넌트를 노드로 대응하고 인터페이스들과 함께 적용하면 컴포넌트들도 3가지 종류로 구성될 수 있다. 이때 중간의 노드는 가지 컴포넌트로 부른다.

트리 컴포넌트 모델을 위해 필요한 인터페이스는 <표 1>에서와 같이 컴포넌트의 종류 별로 구현해할 인터페이스가 다르다. 루트 컴포넌트는 최상위의 컴포넌트로 하위의 컴포넌트만 연결된다. 따라서 하위를 위한 인터페이스(InterfaceForLower)만 필요하다. 잎 컴포넌트는 최하위 컴포넌트로 상위의 컴포넌트만 연결되기 때문에 트리 구조를 이루는 상위를 위한 인터페이스(InterfaceForHigher)와 서비스 로직을 위한 인터페이

〈표 1〉 컴포넌트 종류 별 구현 인터페이스

컴포넌트	구현 인터페이스
루트 (Root)	InterfaceForLower
잎 (Leaf)	InterfaceForHigher, InterfaceForComponent
가지 (Branch)	InterfaceForHigher, InterfaceForLower, InterfaceForComponent

스(InterfaceForComponent) 두 가지가 필요하다. 가지 컴포넌트는 상위와 하위에 모두 연결되므로 상위와 하위를 위한 인터페이스가 모두 필요하다.

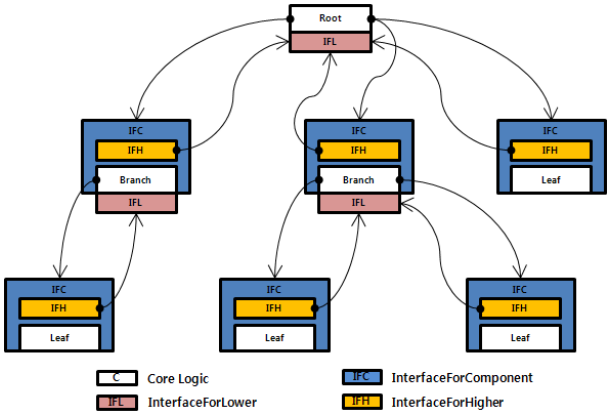
3.1.2 컴포넌트들의 연결 구조

이들 인터페이스를 사용한 전체적인 트리 컴포넌트 모델의 연결 구조는 (그림 3)에서와 같이 표현 될 수 있다. (그림 3)의 각 구성을 살펴보면 Core logic은 컴포넌트가 제공할 계산과 같은 서비스 로직이다. 트리 컴포넌트 모델에서 기능들을 IFC 인터페이스로 랩핑하고 상위 컴포넌트에게 IFC의 레퍼런스를 제공하여 컴포넌트의 서비스 로직들을 사용할 수 있도록 한다. 그리고 IFL 인터페이스는 자신이 포함하는 하위의 컴포넌트에서 상위에 간접 메시지 전달과 트리 구조 연결을 위해 사용되는 인터페이스로 하위 컴포넌트는 상위 컴포넌트의 IFL 레퍼런스를 사용하여 메시지 전달을 한다. IFH는 IFC 인터페이스에 포함되어 간접 메시지 전달과 트리 구조 연결을 위한 인터페이스로 상위컴포넌트는 하위 컴포넌트의 IFC 레퍼런스를 사용하여 IFH의 레퍼런스를 얻는다.

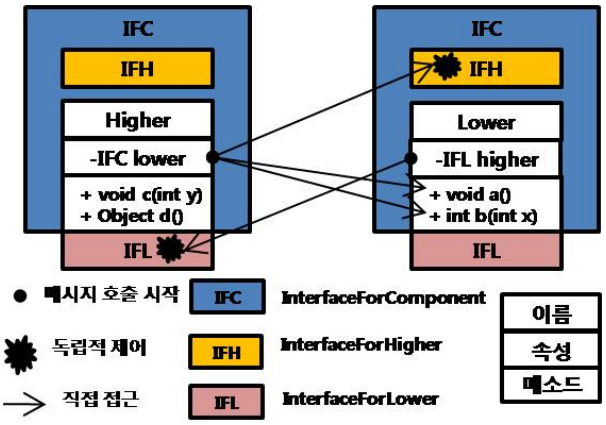
트리 컴포넌트 모델은 상위와 하위가 Linked List와 같이 연결 되는 구조를 가진다. 따라서 트리의 계층 구조상 상위와 하위로 구분되지만 실재로는 상위 컴포넌트가 하위컴포넌트의 인터페이스의 레퍼런스를 가지고 있어 상위가 하위를 포함하는 랩핑 구조를 가진다. 하위도 상위의 레퍼런스를 가지고 있으나 이는 메시지 전달을 위한 커넥터 역할을 하는 인터페이스의 레퍼런스일 뿐이다. 따라서 랩핑의 관점에서 본다면 직접 연결된 내부와 외부간의 통신만이 가능한 컴포넌트로 상위나 하위가 아닌 외부와 내부로 볼 수 있으나 트리 컴포넌트 모델은 (그림 2)에서와 같이 컴포넌트를 아래에서 위로 쌓아가며 랩핑하기 때문에 상위와 하위로 구분하여 설명할 것이다.

3.2 하이브리드 메시지 전달

하이브리드 메시지 전달은 트리 컴포넌트 모델에서 컴포넌트를 연결하는 핵심적인 메커니즘이다. 이는 IFC, IFH, IFL 인터페이스로 계산과 같은 서비스 로직을 랩핑하고 컴포넌트 사이에 메소드 호출, 상호작용, 의사소통 등의 제어들은 모두 이들 3가지 인터페이스의 레퍼런스를 통해서만 이루어진다. 이는 Exogenous 커넥터에서 모든 제어가 발생하고 수행 되는 것과 같이 제어를 계산과 분리한다 하지만 Exogenous 커넥터가 컴포넌트와 분리되어 하나의 객체로 생성되는 반면에 하이브리드 메시지 전달은 커넥터 역할을 하는 인터페이스들과 컴포넌트가 하나의 객체로 생성된다.



(그림 3) 트리 컴포넌트 연결 구조



(그림 4) 컴포넌트 사이의 하이브리드 메시지 전달

또한 하이브리드 메시지 전달을 통해 연결된 컴포넌트들은 트리 구조를 가지므로 컴포넌트들 사이에 순환 호출이 근본적으로 발생하지 않는다.

3.2.1 하이브리드 메시지 전달 메커니즘

(그림 4)과 같이 하이브리드 메시지 전달은 IFC, IFH, IFL 인터페이스를 통한 엄격한 가시성의 제약을 가진다. IFC는 연결된 하위 컴포넌트의 IFH를 포함하는 IFC의 레퍼런스를 가지고 있어 하위에서 공개하는 모든 서비스 로직들에 직접 접근이 가능한 직접 메시지 전달을 위한 인터페이스이다. 이외에 IFC에 포함된 IFH와 하위에게 상위의 가시성을 제공하는 IFL은 트리 구조를 따라 메시지 전달시 사용하는 인터페이스이다. 이들도 인터페이스 레퍼런스를 통해 메소드에 직접 접근이 가능하다. 하지만 이 메소드는 상황에 따라 직접 메시지 전달을 할 수도 있고 간접 메시지 전달을 할 수도 있다. 이것은 (그림 4)에서 표현된 독립적 제어로 키 값과 메시지를 인자로 전달하고 전달받은 키 값에 따라 전달 받은 메시지를 처리하는 방식으로 사용된다.

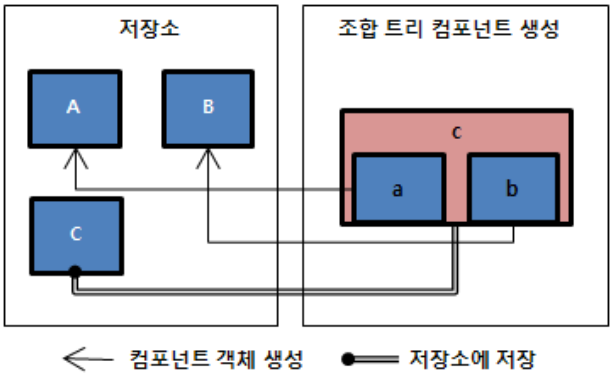
독립적 제어는 트리 컴포넌트를 독립 적인 컴포넌트로 사용되도록 하는 핵심적인 구현 로직이다. 트리 컴포넌트는 자신이 필요한 컴포넌트를 객체로 내부에 포함하기 때문에

대부분의 경우에 직접 접근을 통해 하위 컴포넌트와 메시지 전달을 하게 된다. 이런 직접 접근들은 컴포넌트가 개발될 때 호출 관계가 정해지기 때문에 시스템의 실행 중에 변경되지 않는다. 하지만 시스템이 실행 시에 발생하는 특정 컴포넌트에서의 이벤트를 다른 컴포넌트에 알릴 필요가 있는 경우(예를 들어 공유하는 데이터가 변경된 경우) 연관된 컴포넌트는 이벤트 상황에 맞는 처리가 수행되어야 한다. 그러나 트리 컴포넌트는 독립적으로 사용 가능하도록 구현되기 때문에 실행 시에 발생한 제어를 어떻게 처리할지는 알 수가 없다. 따라서 트리 컴포넌트는 전달 받은 키 값에 따라 미리 수행될 제어를 정의한다.

예를 들어 파일을 선택하는 컴포넌트는 자신이 가진 상위 또는 하위의 레퍼런스를 사용하여 독립적 제어를 수행하는 메소드에게 키 값 (키 값 예 : “CurrentFile”)과 함께 메시지 (메시지 예 : File 객체)를 전달하면 상위 또는 하위의 메소드는 자신이 전달 받은 키 값에 따라 처리할 수도 있고 하지 않을 수도 있다. 또는 자신이 처리하지 않고 전달할 때 하위로부터 메시지를 받은 경우에 상위로 전파하고 상위로부터 메시지를 받은 경우에 하위로 전파하게 된다. 이는 자신이 가진 상위 또는 하위의 레퍼런스를 통해 수행하는 간접 메시지 전달이다. 결과적으로 하이브리드 메시지 전달은 직접 연결된 상위와 하위 컴포넌트 사이에 메시지 전달시 직접 메시지 전달이 가능하고 이외에는 간접 메시지 전달만이 가능하다. 독립적 제어와 관련된 소스 코드는 구현의 예에서 구체적으로 확인 할 수 있다.

3.3 구현 방법

트리 컴포넌트는 앞에서 컴포넌트의 종류에 따라 인터페이스를 구현하여 개발된다. 특히 조합 컴포넌트인 가지와 루트는 (그림 5)에서의 C 컴포넌트들과 같이 바이너리 형태로 저장소에 저장된 필요한 컴포넌트들 객체들을 랩핑한다. 이렇게 생성된 조합 트리 컴포넌트는 내부적으로 다른 컴포넌트와 연결되지만 트리의 각 노드들이 언제나 동일한 방식으로 취급되는 것과 같이 이들 또한 재사용 시에는 내부의 조합 구조와 관계없이 트리 컴포넌트 모델의 인터페이스를 가지는 또 다른 트리 컴포넌트로 사용될 뿐이다. 따라서 저장소에는 이전에 저장되어 있던 컴포넌트들과 동일한 트리 컴포넌트로 저장된다. 이와 같이 사용되기 위해 트리 컴포넌트의 구체적인 인터페이스와 컴포넌트들의 구현 방법 그리고 이들 사이의 조합의 예를 제시한다.



(그림 5) 트리 컴포넌트의 생성과 저장

3.3.1 인터페이스의 구현

트리 컴포넌트는 3가지 인터페이스를 사용한다. 이들은 <표 2>와 같이 각각 특정한 목적을 가지는 인터페이스들이다. 본 논문에서는 JAVA언어를 사용하여 각각의 정의를 구현 할 것이다.

InterfaceForLower는 목적에 따라 2가지 메소드가 필요하다. 이는 (그림 6)과 같이 구현 된다.

- registerAComponent(InterfaceForHigher lower) - 하위 컴포넌트에서 실행되는 메소드로 하위의 InterfaceForHigher 레퍼런스를 전달받아 상위에 등록한다. 이것은 자신의 상위에서 하위 컴포넌트를 연결할 때 받은 상위 객체를 사용하여 호출 된다. 그리고 이 메소드는 내부적으로 자신의 하위 컴포넌트들의 리스트를 유지하여 다수의 하위 컴포넌트가 등록될 수 있도록 한다.
- passAMessageFromLower(Object key, Object message) - 하위에서 트리를 따라 전달된 메시지를 처리하기위해 사용하는 메소드이다. 상위의 컴포넌트에게 메시지를 전달하는 유일한 방법으로 하위 컴포넌트에서 이벤트 호출과 같은 동적인 제어 발생 시 이를 상위로 전파할 필요가 있을 때 호출 되어 메시지 전달을 수행한다. 이는 하위에서 연결된 상위 객체의 레퍼런스를 사용해 호출 할 수 있다. 상위는 하위로부터 전달 받은 메시지의 key 값에 따라 전달된 message를 적절히 처리한다. 이는 책임의 사슬 패턴(Chain of Responsibility) [14]과 같이 사용된다.

InterfaceForHigher 또한 목적에 따라 2가지 메소드가 필요하다. 이는 (그림 7)과 같이 구현 된다.

〈표 2〉 각 인터페이스의 목적

인터페이스	목적
InterfaceForLower	1. 하위를 상위와 연결 2. 하위에서 전달된 메시지 처리
InterfaceForHigher	1. 상위를 하위와 연결 2. 상위에서 전달된 메시지 처리
InterfaceForComponent	1. 컴포넌트의 서비스 로직 호출

```
public interface InterfaceForLower {
   public void registerAComponent(InterfaceForHigher lower);
   public void passAMessageFromLower(Object key, Object message);
}
```

(그림 6) InterfaceForLower의 구현

```
public interface InterfaceForHigher {
   public void setHigher(InterfaceForLower higher);
   public void passAMessageFromHigher(Object key, Object message);
}
```

(그림 7) InterfaceForHigher의 구현

- setHigher(InterfaceForLower higher) - 상위 컴포넌트에서 자신을 인자로 전달하여 상위 컴포넌트를 하위와 연결하는 메소드이다. 이 메소드는 하위의 인스턴스를 사용하여 호출하는 메소드로 제어를 위한 자신의 InterfaceForLower 레퍼런스를 하위에 전달한다. 이때 하위에서는 상위를 반드시 하나만 가지도록 private 객체(higher)에 상위를 저장하고 이를 사용하여 registerAComponent(InterfaceForHigher lower)를 호출할 때 자신의 InterfaceForHigher 레퍼런스를 전달한다. 즉 상위에서 이 메소드 호출 한번으로 컴포넌트간의 트리 구조 연결이 완성된다.
- passAMessageFromHigher(Object key, Object message) - 상위에서 트리를 따라 전달된 메시지를 처리하기위해 사용하는 메소드이다. 이것은 상위 컴포넌트에서 이벤트 호출과 같은 동적인 제어 발생 시 이를 하위로 전달할 필요가 있을 때 호출 하고 key 값에 따라 적절한 제어를 수행한다. 특히 하위의 컴포넌트들에게 전달 할 경우 책임의 사슬 패턴(Chain of Responsibility) [14]과 같이 사용된다.

InterfaceForComponent는 상위에서 하위의 서비스 로직을 직접 호출하는 서비스 인터페이스이다. 이것은 서비스 로직만을 위한 인터페이스로 (그림 8)과 같이 각 컴포넌트의 서비스 로직에 따라 외부에 공개 할 메소드가 다르게 정의된다. 따라서 컴포넌트 마다 다른 메소드가 정의되므로 실제 인터페이스는 각 컴포넌트의 Core Logic의 "이름"을 사용하여 "InterfaceFor이름"으로 명명하여 구분한다. 이 인터페이스가 다른 트리 컴포넌트의 인터페이스와 다른 점은 상위에서 하위 컴포넌트를 하나의 레퍼런스를 사용하여 처리하기 위해 InterfaceForHigher를 포함한다. 이를 통해 상위는 InterfaceForComponent의 레퍼런스만으로 하위의 객체와의 연결을 위한 레퍼런스(IFH)도 얻을 수 있게 된다.

3.3.2 컴포넌트들의 구현

트리 컴포넌트들은 루트, 잎, 가지 3가지가 존재한다. 이들 각각은 구현할 인터페이스가 다르다. (그림 9)에서 (a), (b), (c)와 같이 구현된다. 그리고 각 Core Logic의 이름을 사용하여 InterfaceForComponent의 이름을 변경하였다. 예를 들어 잎 컴포넌트의 Core Logic의 이름이 "Leaf"이므로 "InterfaceForLeaf" (Leaf의 IFC)를 구현한다. 또한 (b), (c)를 자세히 살펴보면 InterfaceForComponent인 InterfaceForLeaf와 InterfaceForBranch는 이미 InterfaceForHigher를 확장하였기 때문에 Core Logic은 InterfaceForHigher를 직접 확장하지 않는다.

(그림 9)의 설계를 기반으로 루트, 잎, 가지 컴포넌트들이 구현된다. 트리 컴포넌트들은 그림과 같이 인터페이스를 랩핑하여 구현되고 이는 2가지의 방법이 존재한다. : (1) 새로운 컴포넌트를 만드는 경우, (2) 트리 컴포넌트가 아닌 컴포넌트를 재사용 하는 경우

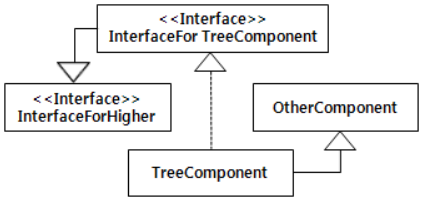
(1)는 컴포넌트의 개발 시 루트, 잎, 가지 중 하나로 개발될 것이다. (그림 9)에서와 같이 컴포넌트의 서비스 로직을 구현하고 각 컴포넌트의 종류에 따라 필요한 인터페이스를 구현 한다. (2)는 트리 컴포넌트가 아닌 컴포넌트를 재사용하여 트리 컴포넌트로 개발하는 것이다. 이는 트리 컴포넌트 모델이 컴포넌트를 인터페이스로 랩핑한 방법이기 때문에 (그림 10)의 예와 같이 어뎁터 패턴 [14, 15]을 사용하여 트리 구조 모델에 맞는 컴포넌트로 변경할 수 있다.

3.3.3 컴포넌트들의 조합의 예

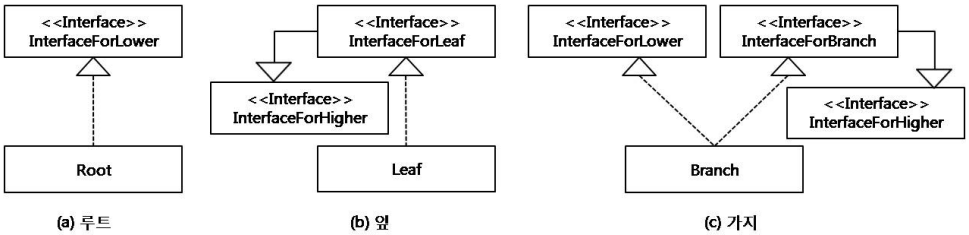
루트, 잎, 가지 컴포넌트들 간의 연결된 조합의 한 가지 예는 (그림 11)과 같다. Root, Branch, LeafA, LeafB의 각 Core Logic들은 인터페이스들을 커넥터로 사용하여 서로 완벽히 분리됨을 확인 할 수 있다. 또한 하이브리드 메시지 전달을 수행하기 위해 직접 하위에 연결된 컴포넌트의 InterfaceForComponent 레퍼런스를 사용하여 직접 메시지 전달을 하고 간접 연결된 상위 또는 하위는 컴포넌트의 InterfaceForLower 또는 InterfaceForHigher의 레퍼런스를 통해 간접 메시지 전달을 한다. 이에 관한 구체적인 예를 들면 LeafB는 InterfaceForHigher 인터페이스를 통해 Branch의 InterfaceForLower의 레퍼런스를 얻어 메시지를 전달한다. 메시지를 전달 받

```
public interface InterfaceForComponent extends InterfaceForHigher {
    // Component의 비즈니스 로직의 인터페이스 작성
    ...
}
```

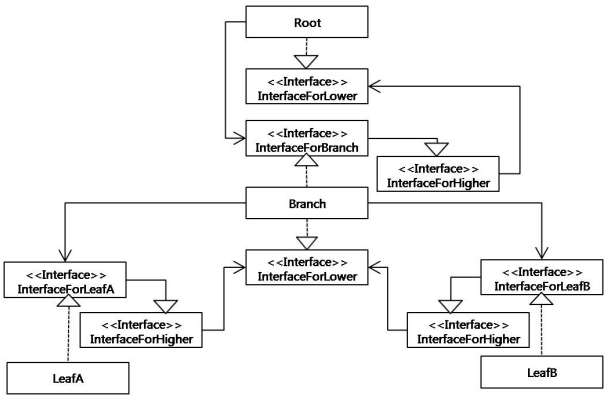
(그림 8) InterfaceForLower의 구현



(그림 10) 어뎁터 패턴을 사용한 잎 컴포넌트



(그림 9) 트리 컴포넌트 모델의 루트, 잎, 가지 컴포넌트



(그림 11) 트리 컴포넌트들 사이의 조합의 예

은 Branch도 자신이 가진 Root의 InterfaceForLower의 레퍼런스를 사용하여 전달 받은 메시지를 Root에 전달한다.

4. 시스템 구현의 예

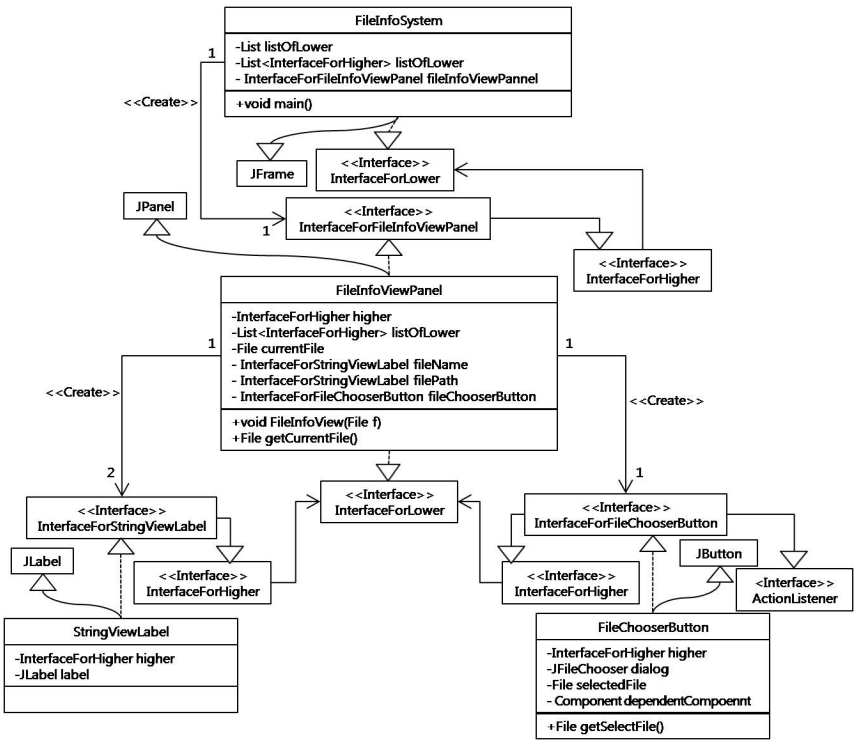
트리 컴포넌트 모델의 간단한 구현의 예로 Swing 컴포넌트 [16]를 트리 컴포넌트로 만들어 구현한다. 구현된 컴포넌트들은 JAVA의 컴포넌트인 Beans [11]로 구현 된다. 이 예는 간단한 파일의 정보를 읽어 그 파일의 경로와 파일 이름을 나타내는 기능을 가진 시스템이다. 구현할 시스템의 조합 구조는 (그림 12)과 같다. 이는 (그림 11)에서의 조합의 예를 적용하여 구현된 시스템으로 FileInfoSystem은 Root와 대응되고 FileInfoViewPanel은 Branch와 대응한다. 그리고 StringViewLabel, FileChooserButton는 LeafA, LeafB와 대응한다. 하위 절을 통해 트리 컴포넌트의 생성과 트리 구조 연결 그리고 하이브리드 메시지 전달의 구체적인 구현 코드의 예를 제시할 것이다.

4.1 트리 컴포넌트의 생성 코드

(그림 12)에서 설계한 컴포넌트 중 가지 컴포넌트인 FileInfo ViewPanel은 트리 구조를 이루는 인터페이스를 위한 모든 코드와 메시지 전달시 독립적 제어를 위한 모든 코드가 들어 있다. (그림 13)는 FileInfoViewPanel의 주요 코드로 JPanel을 어뎁티로 어뎁터 패턴이 적용된 가지 컴포넌트이다. 내부적으로 FileChooserButton과 StringViewLabel을 생성한다. 이때 FileInfoViewPanel은 FileChooserButton이 File 타입의 데이터를 인자로 받아야만 생성할 수 있기 때문에 마찬가지로 File 타입의 데이터를 전달받아야만 생성이 가능하다. 이렇게 생성된 컴포넌트들은 생성과 동시에 setHigher 메소드를 호출하여 트리 구조 연결을 수행한다.

4.1.1 각 메소드의 구현

passAMessageFromHigher 메소드는 "PresentFileInfo"라는 키 값을 전달 받을 경우 수행하는 서비스 로직이 존재한다. 이는 독립적 제어가 발생하는 것으로 자신이 어떤 키 값을 전달 받을 경우 어떤 서비스가 수행되는지를 컴포넌트 명세를 통해 알려야한다. 이 명세는 소스코드 수준의 표현이 될 것이다. (예 : if(key = "PresentFileInfo") then presentFileInfo ((File)message)) 이를 통해 컴포넌트들은 재사용 시 트리 구조를 사용한 메시지 전달이 가능하게 된다.



(그림 12) FileInfoSystem 조합 구조의 설계

```
public class FileInfoViewPannel extends JPanel implements
        InterfaceForLower, InterfaceForFileInfoViewPannel {
                ...
    public FileInfoViewPannel(File f) {
        listOfLower = new ArrayList();
        currentFile = f;
        fileName = new StringViewLabel();          컴포넌트
        fileName.setHigher(this);                  생성 및 연결
        filePath = new StringViewLabel();
        filePath.setHigher(this);
        fileChooserButton = new FileChooserButton(f);
        fileChooserButton.setHigher(this);
                ...
    }

    public void setHigher(InterfaceForLower higher) {
        this.higher = higher;
        higher.registerAComponent(this);
    }

    public void passAMessageFromHigher(Object key, Object message) {
        if (key.equals("PresentFileInfo")) {
            presentFileInfo((File) message);
        }                                          InterfaceForHigher
    }                                              구현 부분

    public void registerAComponent(InterfaceForHigher lower) {
        listOfLower.add(lower);
    }                                              독립적 제어 구현 부분

    public void passAMessageFromLower(Object key, Object message) {
        if (key.equals("SelectingFile")) {
            presentFileInfo((File) message);
            higher.passAMessageFromLower("SelectingFile", message);
        }                                          InterfaceForLower
    }                                              구현 부분
                ...
}
```

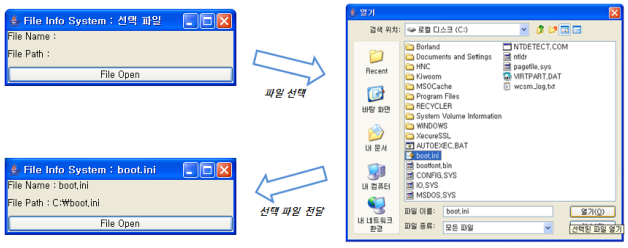
(그림 13) FileInfoViewPanel의 주요 코드

passAMessageFromLower 메소드는 하위로부터 "SelectingFile"이라는 키 값을 전달 받을 경우 수행하는 서비스 로직이 존재한다. 이는 이미 FileChooserButton의 명세에 따라 발생하는 이벤트에 적절하게 대응하는 InterfaceForLower 인터페이스의 메소드이다. 즉, 상위에 공개되지 않는 메소드이다. 하지만 내부 로직에서 수행되는 higher.passAMessageFrom Lower("SelectingFile", message); 코드는 책임 사슬 패턴[14]과 같이 사용된 간접 메시지 전달 방법으로 하위 컴포넌트에서도 같은 방법으로 발생한 이벤트를 전달한다. 그러므로 FileInfoViewPanel의 상위의 관점에서 이는 이벤트를 전달하는 컴포넌트이다. 따라서 컴포넌트 내부에서 어떤 경우에 이벤트가 발생되고 어떤 키로 어떤 값이 전달되는지를 소스코드 수준으로 명세화해야 한다. 이 경우 이미 하위의 컴포넌트의 명세를 통해 사용하였기 때문에 어떤 이벤트에 의해 이와 같은 메시지 전달이 발생했는지 알고 있으므로 하위의 명세를 재사용 할 수 있다. 그리고 이 독립적 제어 로직은 presentFileInfo((File) message); 수행하여 내부적으로 fileName과 filePath 레퍼런스를 통해 직접 메시지 전달도 수행한다.

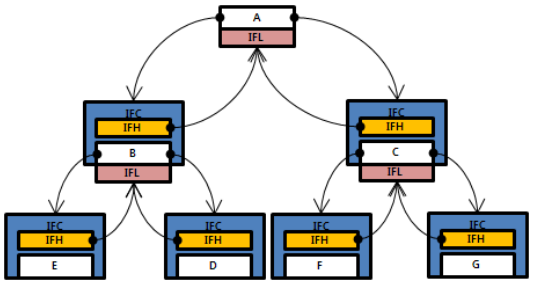
코드에서 보여지는 Core Logic인 FileInfoViewPanel은 각 인터페이스의 구현을 포함하고 있어 현재의 JAVA와 같은 객체지향 언어에서는 FileInfoViewPanel 형 레퍼런스를 생성하여 사용할 수도 있다. 하지만 이는 트리 컴포넌트 모델의 규칙을 위반하는 것이다. 따라서 다른 컴포넌트의 메소드 호출과 컴포넌트 간의 상호 작용 등과 같은 모든 제어를 수행할 때, 반드시 인터페이스 레퍼런스를 사용하해야만 한다.

4.1.2 시스템의 실행

(그림 14)은 FileInfoSystem의 실행 과정이다. 이를 통해 컴



(그림 14) FileInfoSystem의 실행 과정



(그림 15) 트리 컴포넌트 모델의 컴포넌트 연결

포넌트간의 제어가 이루어지는 과정을 코드의 관점에서 살펴보면 FileChooserButton 컴포넌트에서 이벤트를 받고 이때 사용자에 의해 선택된 파일을 FileChooserButton의 selectedFile 인자에 저장한 뒤 상위 컴포넌트의 레퍼런스인 higher를 사용하여 higher.passAmessageFromLower("SelectingFile", selected File);과 같이 호출하면 (그림 13)의 passAMessageFromLower 메소드가 호출되고 앞에서의 구현 설명과 같이 직접 메시지 전달과 간접 메시지 전달이 수행 된다.

5. 비교 평가

트리 컴포넌트 모델은 Ex 모델에서 컴포넌트 사이의 연결과 계층의 증가 시 발생하는 시스템 구성과 메시지 전달을 중재하는 객체의 수가 증가하는 문제를 개선한다. 이를 증명하기위해 앞에서의 Ex 모델의 연결 구조를 묘사한 (그림 1)을 트리 컴포넌트 모델에 적용한 연결 구조를 묘사한 (그림 15)과 서로 비교할 것이다.

Ex 모델은 제어를 커넥터로 분리하여 객체로 생성하고 컴포넌트는 계산만을 수행하는 객체로 분리하여 생성하므로 커넥터 중심의 연결 관점의 계층 구조를 가진다. 반면 트리 컴포넌트 모델은 제어와 계산이 하나의 객체로 생성되고 인터페이스 레퍼런스를 사용하여 제어와 계산을 분리하므로 컴포넌트 사이의 연결 관점 중심의 계층 구조를 가진다. 그러므로 같은 시스템을 개발했을 때 트리 컴포넌트 모델은 커넥터 객체가 존재하지 않으므로 Ex 모델보다 시스템을 구성하는 객체의 수가 더 적게 된다.

또한 컴포넌트 사이의 메시지 전달을 중재하는 객체 수의 억제를 확인하기 위해 <표 3>은 트리 컴포넌트 모델과 Ex 모델에서의 E에서 전달할 수 있는 모든 구간에 따라 그 사이를 중재하는 객체의 수를 조사한 것이다.

〈표 3〉 메시지 전달을 중재하는 객체의 수 비교 표

구간	트리 컴포넌트 모델	Ex 모델
E→D	1	3
E→B	0	3
E→A	1	5
E→C	2	6
E→F	3	7
E→G	3	7

<표 3>에서 특별한 구간은 E→B이다. 이 구간은 트리 컴포넌트 모델에서 컴포넌트 사이가 직접 연결되어 있다. 따라서 중재하는 객체가 존재하지 않고 직접 메시지 전달을 한다. 하지만 Ex 모델은 이 구간에서 InvE, S1, InvB와 같은 3개의 객체가 중재를 하고 있다. 또한 Ex 모델에서 둘 사이의 중재 객체의 수가 최소가 되는 E→D 구간에도 컴포넌트가 연결 될 때 반드시 하나의 호출 커넥터가 필요하므로 두 컴포넌트 사이의 메시지 전달시 언제나 3개 이상의 중재 객체가 존재하게 된다. 따라서 시스템을 구성하는 객체의 수와 <표 3>에서 확인 할 수 있는 중재 객체의 수를 통해 시스템 구현 시, 트리 컴포넌트 모델이 Ex 모델보다 시스템 구성과 메시지 전달을 중재하는 객체의 수를 감소시키는 것을 증명한다.

6. 논의 및 결론

트리 컴포넌트 모델은 컴포넌트들을 객체로 재사용한다. 생성된 객체는 반드시 하나의 상위 컴포넌트만 가질 수 있다. 즉, 다른 컴포넌트에서 동일한 컴포넌트를 사용해야 한다면 그곳에서 사용할 컴포넌트를 새로 생성하여 재사용 한다. 만약 내부의 컴포넌트가 공유되어 사용되어야 한다면 공유해야 할 컴포넌트를 서비스로 다른 컴포넌트에게 제공할 수 있을 것이다. 또한 트리 컴포넌트 모델에서 사용한 랩핑 방식의 컴포넌트 조합은 상위의 컴포넌트가 하위의 컴포넌트에게 강한 의존성을 가진다. 이는 하위의 컴포넌트가 없이는 개발이 불가능하고 상위 컴포넌트가 하위 컴포넌트에 강한 의존도를 가짐을 의미한다.

트리 컴포넌트 모델은 컴포넌트간의 조합을 위해 하이브리드 메시지 전달 메커니즘을 사용한다. 이는 컴포넌트 사이의 연결 구조가 언제나 트리 형태의 계층적 구조를 가진다. 따라서 컴포넌트 사이의 조합이 추적가능하며 예측 가능하고 모든 컴포넌트들은 상위에서 연결 메소드(setHigher)를 호출하는 동일한 방법으로 조합된다. 트리 컴포넌트 모델의 특징을 정리하면 아래와 같다.

1. 컴포넌트들 간의 조합을 반복하여도 계속해서 트리 구조를 이루어 연결이 된다.
2. 아래에서부터 위로 랩핑하여 조합을 한다.
3. 트리 컴포넌트는 동일한 방식으로 조합된다.

본 논문이 제안한 트리 컴포넌트 모델은 계층 구조를 가지는 새로운 모델로서 제어와 계산의 분리를 통해 컴포넌트 사이의 의존성을 낮추고 하이브리드 메시지 전달을 통해 시스템을 구성하여 메시지 전달을 중재하는 객체의 수를 감소시킨다. 또한 인터페이스 랩핑을 사용한 방식으로 다른 방법의 컴포넌트를 재사용하기 용이하다. 이는 트리라는 익숙한 계층 구조를 컴포넌트에 접목한 새로운 시도이다. 아직 트리 모델의 구체적인 효용성을 예측할 수 없으나 본 논문에서 제안한 트리 컴포넌트 모델이 아닐지라도 트리 구조로 컴포넌트를 조합하는 것이 CBD의 근본을 가장 잘 따르는 조합 방법으로 사료된다. 특히 데스크탑 환경[17]에서의 소프트웨어와 같이 단일 환경에서 사용되는 컴포넌트 개발에 트리 컴포넌트 모델을 쉽게 적용할 수 있을 것이다.

따라서 앞으로의 연구 과제로 데스크탑 환경에서 트리 컴포넌트 모델의 실용성을 평가할 것이다. 이는 실제 소프트웨어 시스템 개발에 적용 가능한 트리 컴포넌트의 개발을 필요로 한다. 그리고 데스크탑 환경에서 사용하는 컴포넌트 모델인 JavaBeans [3, 11]와 트리 컴포넌트 모델을 비교하여 타당성을 검증할 것이다.

참 고 문 헌

[1] C. Szyperski, D. Gruntz, and S. Murer, 'Component Software: Beyond Object-Oriented Programming,' Second Edtion, ADDISON-WESLEY, 2002.

[2] X. Xiaoqin, X. Peng, L. Juanzi, and W. Kehong, "A Component Model for Designing Dynamic GUI," Parallel and Distributed Computing, Applications and Technologies, PDCAT'2003. Proceedings of the Fourth International Conference, pp. 136-140, 2003.

[3] K.-K. Lau and Z. Wang, "Software Component Models," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol.33, No.10, pp.709-724, 2007.

[4] K.-K. Lau and Z. Wang, "A Taxonomy of Software Component Models," Proc. 31st Euromicro Conf. Software Eng. And Advanced Applications (SEAA '05), pp.88-95, 2005.

[5] K.-K. Lau, P. Velasco Elizondo, and Z. Wang, "Exogenous Connectors for Software Components," In: Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Szyperski, C.A., Wallnau, K. (eds.) CBSE 2005. LNCS, Vol.3489, pp.90-106, Springer, Heidelberg, 2005.

[6] K.-K. Lau, F. M. Taweel, V. Ukis, P. Velasco, and Z. Wang, "A Component Model for Separation of Control Flow from Computation in Component-Based Systems," Electronic Notes in Theoretical Computer Science 163 (2006), 57-69, ScienceDirect, 2006.

[7] K.-K. Lau, and F. M. Taweel, "Towards Encapsulation Data in Component-Based Software Systems," CBSE 2006, LNCS 4063, pp. 376-384, Springer, Heidelberg, 2006.

[8] K.-K. Lau, L. Ling, and P. Velasco Elizondo, "Towards Composing Software Components in Both Design and Deployment

Phases," CBSE 2007, LNCS 4608, pp.274-282, Springer, Heidelberg, 2007.
[9] E. Gossett, 'Discrete Mathematics with Proof,' Prentice Hall, 2003.
[10] B. Long and P. Strooper, "A Classification of Concurrency Failures in Java Components," Parallel and Distributed Processing Symposium, Proceedings. International, 8 pp, 2003.
[11] R. Englander. 'Developing Java Beans,' O'Reilly & Associates, 1997.
[12] 임윤선, 김명, 정승남, 정안모, "컴포넌트 재사용을 지원하는 컴포넌트 모델 및 프레임워크," 한국정보과학회, 2007.
[13] H. Washizaki, and Y. Fukazawa, "A Model-View Separation Structure for GUI Application Components," Information Technology. Coding and Computing, ITCC 2005. International Conference Volume 2, pp.359-364, Vol.2, 2005.
[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, 'Design Patterns,' Addison-Wesley, 1995.
[15] E. Freeman, K. Sierra, B. Bates, 'Head First Design Patterns,' O'REILLY, 2004.
[16] The Swing Tutorial, 'http://java.sun.com/docs/books/tutorial/uiswing/,' Sun Microsystems, 2007.
[17] K.-K. Lau, V. Ukis, "A Study of Execution Environments for Software Components," CBSE 2007, LNCS 4608, pp. 107-123, Springer, Heidelberg, 2007.

허 제 민

e-mail : hjm1980@kw.kyungwon.ac.kr
2005년 경원대학교 컴퓨터공학과(학사)
2009년 경원대학교 전자계산학과 졸업 예정(석사)
관심분야 : 소프트웨어 공학, CBD, SOA, 소프트웨어 컴포넌트

김 지 홍

e-mail : wiskjh@kyungwon.ac.kr
1974년 경희대학교 공과대학 전자공학과 (학사)
1982년 미국 California State University (Fullerton) 대학원 전자계산학과 (석사)
1995년 경희대학교 대학원 전자계산공학과(박사)
1982년~1989년 미국 Interpac Software, 소프트웨어 엔지니어
1989년~현 재 경원대학교 IT대학 소프트웨어학부 교수
관심분야 : 소프트웨어 공학, UML, 소프트웨어 컴포넌트, 소프트웨어 프로덕트라인 공학