컴포넌트 결합 명세서에 기반한 컴포넌트 결합 모델

백 경 원[†] · 박 성 은[†] · 이 정 태[†] · 류 기 열[††]

요 약

현재 대부분의 컴포넌트 기반 개발 환경 들이 특정 형태 컴포넌트와 이를 위한 프레임웍 기능 제공을 위주로 하는데 대하여, 좀 더 효과적인 컴포넌트 기반 개발 환경의 구축을 위해서는 재귀적 컴포넌트 결합 및 다양한 결합형태의 지원, 그리고 다중 티어 아키텍처를 지원할 수 있는 방안의 필요성이 제기되고 있다. 이의 해결 방안의 하나로 본 논문에서는 재귀적 컴포넌트 결합을 지원할 수 있으며, 컴포넌트간 상호 동작 조건을 기술할 수 있도록 확장된 컴포넌트 결합 명세의 기술 방안 및 이를 이용한 컴포넌트 결합 모델을 제안하였다. 제안된 결합 명세서는 기존의 컴포넌트 기술 방법들에서 지적되고 있는 컨트랙트로서의 부족한 기능을 보완하고 있을 뿐만 아니라 다양한 형태의 컴포넌트간 결합을 일관된 형태로 지원 함으로써 다중 티어(Multi-Tier) 개념을 지원하는 컴포넌트 아키텍쳐의 구현 및 이를 위한 컴포넌트 결합 도구의 구현에 활용될 수 있다.

A Component Composition Model based on Component Composition Specification

Kyung-Won Baek[†] · Sung-Eun Park[†] · Jung-Tae Lee[†] · Ki-Yeol Ryu[††]

ABSTRACT

Today's wide variety of component-based development environments supports the component framework that can be used only for the specific type of components. And many researches have shown that it is necessary for the component-based development environment to support recursive component composition, various kinds of component composition patterns and the multi-tier component architecture for the real benefits of software component composition. In this paper we propose the component composition specification which can not only specify the interaction contracts between components but also supports recursive component composition, and we also propose the component composition model based on this component composition specification. The proposed component composition specification can express the contractual properties that existing component specification techniques cannot specify, and it can be also used to implement the component architectures with multi-tier concept and the tool for component composition through supporting the various kinds of component composition patterns.

키워드 : 컴포넌트 결합 명세서(Component Composition specification), 재귀 컴포넌트 결합(Recursive component composition), 다중 티어 아키텍처(Multi-tiered architecture), 컴포넌트 결합 도구(Component composition Tool), 컴포넌트 프레임웍(Component framework), 컴포넌트 아키텍처(Component architecture), 컴포넌트 기술(Component specification)

1. 서 론

최근 본격적인 컴포넌트 기반 시스템의 구축을 위해서는 컴포넌트 시스템의 아키텍처가 컴포넌트 프레임웍 자체를 일관된 결합방식에 의하여 결합된 컴포넌트로 간주하는 다중 티어 개념(N-tiered component architecture)의 아키텍처를 지원해야 하며[4, 8], 이를 위해서는 컴포넌트 배치(component deployment), 프레임웍 배치(framework deployment), 단순 결합(simple composition), 이질간의 결합(heterogeneous composition), 프레임웍 확장(framework extension), 그리고 컴포넌트 어셈블리(component assembly)등의 다양한 컴포넌트 결합을 허용해야 한다는 제안이 제기되고 있다[2].

다양한 컴포넌트 결합을 제공하기 위해서는 컴포넌트의 명세 부분(specification)과 구현 부분을 완전히 분리함으로써 제 3자가 구현의 세부를 고려하지 않고 잘 정의되어 있는 컴포넌트의 명세 부분만을 이용하여 컴포넌트들을 결합할 수 있어야 하며, 이에 바탕을 둔 컴포넌트 시스템 아키텍처 및 컴포넌트 결합 도구가 지원되는 응용 프로그램 개발환경이 제공되어야 한다[4].

이를 위한 컴포넌트 명세 방법에서는 주로 기존의 객체지향 방식에서 사용되는 인터페이스와 기능적 인터페이스의 사용에 대한 상호 문맥적인 조건을 나타내는 선조건(precondition) 및 후조건(postcondition) 등에 기반한 컴포

† 정 회 원 : 아주대학교 대학원 정보 및 컴퓨터 공학부
†† 종신회원 : 아주대학교 정보 및 컴퓨터 공학부 교수
논문접수 : 2001년 10월 5일, 심사완료 : 2001년 12월 17일

넌트 명세방법에 대하여 주로 연구가 많이 진행되어 왔다 [2,4,5]. 이렇게 명세된 컴포넌트의 명세 부분은 컴포넌트의 결합 시 상호간 만족되어야 할 결합 조건을 명세하고 있다는 측면에서 컨트랙트(contract)라고 부르기도 한다. 그러나 기존의 명세 방법에서는 대부분 특정 컴포넌트의 기능 및 이 기능의 사용 조건은 자세하게 표현할 수 있지만 컴포넌트들간의 상호동작(Interaction)에 관한 조건은 잘 표현하지 못하고 있다.

기존의 컴포넌트 명세방법의 또 하나의 단점으로는 컴포넌트 명세 자체가 컴포넌트의 결합을 효과적으로 지원하지 못하고 있다는 점이다. 대부분 프로그래머가 컴포넌트 명세를 읽고 해석하여 코딩으로 컴포넌트의 결합작업을 수행해야 하는 것이 일반적이다. 그러나 이러한 방법은 결합 작업이 복잡하고 오류를 일으키기 쉬우며, 무엇보다도 컴포넌트에 기반한 소프트웨어 개발의 철학에 부응하지 못하는 방식이다. 또한 결합작업을 효과적으로 지원하는 도구의 개발이 어렵다. 이러한 문제를 해결하기 위한 방법으로는 컴포넌트 결합을 결합 명세서(composition specification)로 작성하는 방법이 있다. 결합 명세서는 일종의 결합 명세언어를 이용하여 결합되어질 컴포넌트들의 명세를 기반으로 컴포넌트를 결합하는 고 수준의 결합 방식이다. 또한 결합명세서로 기술된 명세수준의 복합 컴포넌트로부터 실행 가능한 복합 컴포넌트를 자동으로 생성하는 것도 가능하다.

결합명세서에 기반한 컴포넌트 결합모델에 관한 연구로는 자바빈즈에 기반한 컴포넌트 결합 언어인 BML(Bean Markup Language)을 이용한 컴포넌트 결합모델[1]이 있다. 이 방법은 자바빈즈 컴포넌트에 제한적으로 적용되며, 또한 결합에 참여하는 컴포넌트들이 자바빈즈 이벤트에 의해서만 연결되며, 다른 상호동작에 관한 제약조건[4]의 명세를 표현하지 못한다는 단점이 있다. 또한 컴포넌트와 컴포넌트의 결합이 새로운 복합 컴포넌트를 만들어내는 재귀적 결합(recursive composition)이 가능하지 않으며, 다른 컴포넌트들이 복합 컴포넌트에 포함되는 제한된 결합을 지원한다. 마지막으로 이 방법은 앞에서 언급한 다양한 컴포넌트의 결합방법을 지원하지 못한다.

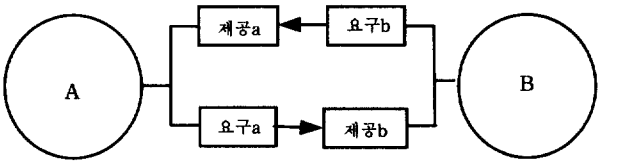
본 논문에서는 기존의 인터페이스와 컴포넌트들간의 상호동작의 조건을 표현할 수 있도록 확장한 명세방법을 제안하고 이러한 명세방법에 기반한 컴포넌트의 결합모델을 제안한다. 제안된 방법에서는 컴포넌트의 결합을 결합 명세서의 형태로 기술하며 이 결합명세서는 복합 컴포넌트가 된다. 따라서 이 결합 명세서로부터 복합 컴포넌트를 자동 생성할 수 있다. 또한 본 논문에서는 결합명세서에 기반한 결합모델이 [2]에서 제안한 다양한 형태의 결합방식을 어떻게 지원하는지를 보여준다.

2절에서는 결합 명세서의 구조를 설명하고, 기존 컴포넌트 명세 방안에서의 문제를 어떻게 보완할 수 있는지를 언급한다. 3절에서는 결합 명세서에 기반한 컴포넌트의 결합 모델을 기술하고 컴포넌트간 다양한 결합에 어떻게 활용될 수 있는지를 설명한다. 4절에서는 결합 명세서의 구현 방안을 제시함으로써 컴포넌트 결합 도구의 구현에 활용될 수 있음을 보여준다. 마지막으로 5절에서는 관련 연구와 비교 분석하고 6절에서 결론을 맺는다.

2. 재귀적 결합을 지원하는 결합 명세서

2.1 결합 명세서 구조

컴포넌트간 상호동작 시 한 컴포넌트는 다른 컴포넌트의 서비스를 필요로 하게 되며 다른 컴포넌트의 필요한 부분을 만족시키기 위하여 서비스를 제공하게 된다. 따라서 컴포넌트의 인터페이스를 기술하는데 있어 각 기능들은 (그림 1)에서 보는 바와 같이 제공 서비스와 요구 서비스로 구분되어질 수 있다.



(그림 1) 제공 서비스와 요구 서비스

컴포넌트 A는 제공 a, 요구 a를 컴포넌트 B는 요구 b, 제공 b를 명세로서 가지게 된다. 이때 두 컴포넌트 A와 B를 결합하게 될 경우 A의 제공 서비스와 B의 요구서비스가 상호 동작하게 되고 A의 요구 서비스와 B의 제공 서비스가 상호 동작하게 된다. 이 경우 두 컴포넌트의 결합은 상호 사용 관계만을 가지게 되며 상호 동작에 대한 조건이 주어지기 어렵다. 만약 A의 명세 부분에 제공 서비스를 사용하는데 있어 선조건, 후조건을 제시하고 있더라도 결합되어 질 수 있는 불특정 컴포넌트를 대상으로 조건을 명세하기 때문에 결합시 발생할 수 있는 모든 경우에 대하여 조건을 제시하기 어렵다. 컴포넌트 결합시 결합에 참여하는 컴포넌트의 명세에 결합에 대한 조건을 기술하기 어려운 이유를 좀더 세분해 보면 다음과 같다.

첫째, 컴포넌트의 개발자는 향후 개발 될 컴포넌트가 어떤 기능을 갖는 복합 컴포넌트의 요소로 결합될 지 예측할 수 없으므로 복합 컴포넌트의 인터페이스가 되어야 하는 기능은 결합에 참여하는 컴포넌트의 인터페이스에 정의되어질 수 없다. 따라서 재귀적 결합에 의해서 생성되는 복합 컴포넌트의 인터페이스는 결합자에 의하여 정의되어야 한다.

둘째, 컴포넌트의 기능 중 어떤 기능이 제공 서비스가 되고 어떤 기능이 요구 서비스가 되어야 하는가는 일부 기능의 경우 컴포넌트 개발 시 결정되어질 수도 있으나

일부 기능은 결합되는 상대 컴포넌트에 따라 변화 될 수 있다. 따라서 컴포넌트의 기능중 요구, 제공 서비스의 정확한 분류 역시 결합자에 의하여 정의되어야 한다.

셋째, 컴포넌트간 결합에 의하여 발생하는 상호동작 조건 역시 각각의 컴포넌트 개발 시 일부 고정되는 조건이 있을 수 있으나 결합되는 상대 컴포넌트에 따라 또는 생성되는 복합 컴포넌트가 제공하고자 하는 기능에 따라 변화할 수 있다.

따라서 본 논문에서는 결합에 참여하는 각 컴포넌트들의 명세와는 구분하여 별도로 결합을 위한 명세를 기술하는 방안을 제시하고자 한다. 이를 위한 명세서 구조에서는 복합 컴포넌트의 인터페이스, 결합에 참여하는 개개 컴포넌트 기능에 대한 제공과 요구 서비스 분류, 결합에 따른 상호동작 조건이 명시적으로 기술되어야 한다.

또한 이러한 기술 내용은 컴포넌트를 결합한 실행 가능한 복합 컴포넌트의 형태로 구현될 수 있어야 하므로 이에 대한 명세 내용은 프로그램에 의하여 해석될 수 있을 만큼 충분히 정형적 이어야 한다. 이와 같은 요구 조건을 충족시키기 위하여 새로운 명세 방안을 제안한다.

(그림 2)는 본 논문에서 제시하고 있는 결합 명세의 구조로서 크게 세 부분으로 구분되어 진다.

- 결합에 참여하는 컴포넌트 목록
- 결합에 참여하는 컴포넌트의 제공 또는 요구서비스 분류
- 복합 컴포넌트에 대한 새로운 서비스 정의
- 결합에 참여하는 컴포넌트의 상호동작 조건

첫 번째 부분에서는 결합에 참여하는 컴포넌트들의 목록을 기술한다. 두 번째 부분에서는 첫 번째 부분에서 명세된 결합에 참여하는 각 컴포넌트의 서비스를 그대로 수용하여 제공 또는 요구 서비스로 분류하여 기술한다. 이 중 일부를 선택하여 복합 컴포넌트 인터페이스의 제공 또는 요구 서비스로 등록(**EXPORT**)하게 되며 이로써 복합 컴포넌트가 제 3자에 의하여 다시 새로운 컴포넌트 결합에 참여 할 수 있다.

```
COMPOSITION CONTRACT component_name
IMPORT import_component_name_list
{ import_component_name
    (* REQUIRES *)
    service_name()      : EXPORT
    service_name()      : EXPORT
    ......
    (* PROVIDES *)
    service_name()      : EXPORT
    service_name()
    ......
} +
```

```
INTERACTION CONSTRAINT
    (* INIT *)
    service_name()
    (* CONSTRAINT *)
    { service_name() :
        { interaction_constraint } +
} +
```

(그림 2) 결합 명세 구조

세 번째 부분에서는 다시 두 부분으로 세분되어지는데 한가지는 복합 컴포넌트로서 동작하기 위한 초기화 부분이다(**INIT**). 초기화 부분은 복합 컴포넌트가 초기에 올바로 시동할 수 있도록 결합에 참여하는 컴포넌트의 서비스 중 일부를 이용하여 초기화 조건으로 명세 할 수 있다. 결합에 참여하는 각 컴포넌트의 명세에는 결합에 참여했을 때 다른 컴포넌트와 상호동작으로 발생할 수 있는 모든 문맥적 의미를 명세하기 어렵다. **INIT** 부분은 상호동작 중 먼저 발생해야 하는 서비스를 명세하는 방법을 제시하고 있다.

다른 한가지는 두 번째 부분에서 명세한 복합 컴포넌트의 제공 또는 요구 서비스에 대하여 결합에 참여하는 컴포넌트들의 서비스를 이용하여 상호동작 조건을 명세 하는 부분(**CONSTRAINT**)이다. 컴포넌트 결합시 상호동작 조건 명세 방법에 대해서는 다양한 방법이 있을 수 있는데 본 논문에서 제시하는 방안에서는 상호동작으로 발생하는 서비스들의 실행 시퀀스(sequence)에 대하여 조건을 명세 함으로써 결합시 유지되어야 할 상호동작 조건을 표현하고 있다. 상호동작 조건을 표현하기 위하여 상호동작 조건을 크게 두 가지로 분류하였다.

- must : 반드시 발생되어야 하는 시퀀스
- must not : 반드시 발생되지 말아야 하는 시퀀스

결합에 참여하는 컴포넌트들간의 상호동작으로 많은 실행 시퀀스가 발생할 수 있는데 이 중 반드시 발생해야 하는 시퀀스(must)와 발생되지 말아야 하는 시퀀스(must not)를 결합 명세 상에 명시함으로써 결합에 참여하는 컴포넌트간의 상호결합 조건을 표현하고 있다.

2.2 결합 명세의 예

본 절에서는 파일 시스템의 서비스를 가지는 Directory 컴포넌트와 디렉터리를 화면에 보여주는 DirectoryDisplay 컴포넌트를 결합 시 사용될 수 있는 결합 명세의 예를 들어 설명하고 있다.

(그림 3)은 Directory 컴포넌트의 인터페이스 명세의 모습을 보여주고 있다.

Directory 컴포넌트는 이 컴포넌트를 사용할 Notifier컴포넌트의 등록과 해제에 대한 서비스와 파일 시스템의 삭제, 추가 변경 사항에 대하여 등록되어 있는 컴포넌트에 이 사실을 알려주는 서비스를 제공하며 각각의 서비스는 서비스

를 사용하기 위한 조건으로 pre와 post 조건을 가진다.

```
DEFINITION Directory ;
   IMPORT Files ;
   TYPE
      Name = ARRAY OF CHAR
      Notifier = PROCEDURE(IN name : Name) ;
   PROCEDURE AddEntry(n : Name ; f : Files.File) ;
      (* pre and post conditions *)
   PROCEDURE RemoveEntry(n : Name) ;
      (* pre and post conditions *)
   PROCEDURE RegisterNotifier(n : notifier) ;
      (* pre and post conditions *)
END Directory.
```

(그림 3) Directory 컴포넌트

(그림 4)는 Directory 컴포넌트에 자신을 등록한 후 Directory 컴포넌트의 변화에 따라 변화된 내용을 전달해 오면 변화된 내용을 화면에 보여주는 DirectoryDisplay 컴포넌트이다.

```
DEFINITION DirectoryDisplay ;
   IMPORT Directory ;
   PROCEDURE Notifier(IN n : Directory.Name) ; (*pre and post
   condition *)
```

(그림 4) DirectoryDisplay 컴포넌트

제 3 자는 위의 두 개의 컴포넌트를 결합하여 사용할 수 있으며 이 때 두 컴포넌트 사이에 상호동작이 이루어지며 이에 대하여 복합 컴포넌트 작성자의 작성 의도에 맞게 조건을 제시할 수 있는 명세가 필요하게 된다.

(그림 5)는 Directory 컴포넌트와 DirectoryDisplay 컴포넌트의 결합 명세의 한 예를 보여 주고 있다. 제 3 자는 두 컴포넌트를 **IMPORT**하여 결합하게 되는데 이때 결합에 참여하는 컴포넌트의 자체 명세는 그대로 계승받게 된다. 즉 위의 두 개의 컴포넌트 명세부분의 pre와 post 조건을 그대로 계승받아 결합 명세의 조건이 동작할 뿐만 아니라 서비스의 요청이 결합에 참여하는 컴포넌트에 이르게 되면 각 컴포넌트의 명세서에 기술되어 있는 pre와 post 조건을 수행하게 된다.

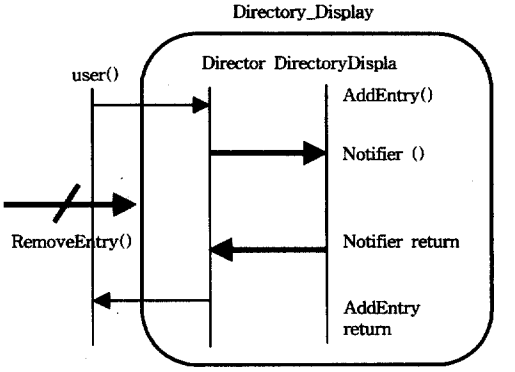
다음으로 IMPORT에 의해 결합에 참여하는 각 컴포넌트의 제공 또는 요구 서비스를 명세하게 되는 데 이중 제 3 자의 복합 컴포넌트의 작성 의도에 알맞은 서비스를 다시 복합 컴포넌트의 제공 또는 요구 서비스로 등록하여 사용하게 된다 **(EXPORT)**. 등록되어 사용되는 서비스들은 재귀적 컴포넌트 결합에 의하여 생성되는 복합 컴포넌트의 제공(AddEntry(), RemoveEntry()) 또는 요구(RegisterNotifier()) 서비스로서 사용된다.

상호동작 조건(INTERACTION CONSTRAINT)의 처음 부분에는 두 컴포넌트를 결합하여 시동시키기 위하여 DirectoryDisplay 컴포넌트가 Directory 컴포넌트에 등록되어 있어야 함을 의미한다. 복합 컴포넌트의 사용자는 우선 **INIT** 에 명세되어 있는 조건을 만족시켜야 한다. 즉 복합 컴포넌트 사용자는 DirectoryDisplay 컴포넌트를 등록하기 위하여 Directory 컴포넌트의 RegisterNotifier() 서비스를 호출하여야 한다.

EXPORT에 의하여 복합 컴포넌트의 제공 서비스로 등록되어 있는 서비스에 대하여 결합에 참여하는 컴포넌트간 상호동작 조건을 명세할 수 있다. 즉 복합 컴포넌트의 AddEntry() 서비스 사용 시 결합에 참여하는 컴포넌트의 상호동작으로 발생할 수 있는 시퀀스 중 서비스가 시작되면 DirectoryDisplay.Notifier() 서비스가 반드시 발생해야 하며 DirectoryDisplay.Notifier() 서비스 후에 Directory.RemoveEntry() 서비스는 발생하지 말아야 함을 의미하고 있다(그림 6). 이는 서비스가 시작되면 엔트리를 추가하는 동안에는 삭제할 수 없음을 의미한다.

```
COMPOSITION CONTRACT Directory_Display
IMPORT Directory, DirectoryDisplay
Directory
   (* REQUIRES *)
      Notifier()
   (* PROVIDES *)
      AddEntry()                    : EXPORT
      RemoveEntry()                 : EXPORT
      RegisterNotifier()
DirectoryDisplay
   (* REQUIRES *)
      RegisterNotifier()            : EXPORT
   (* PROVIDES *)
      Notifier ()
INTERACTION CONSTRAINT
   (* INIT *)
   DirectoryDisplay .RegisterNotifier()
   (* CONSTRAINT *)
   Directory .AddEntry() :
      (* MUST *)      DirectoryDisplay. Notifier ()
      (* MUST NOT *)
         DirectoryDisplay. Notifier () -> Directory .RemoveEntry ()
   Directory .RemoveEntry() :
      (* MUST *)      DirectoryDisplay. Notifier ()
      (* MUST NOT *)
         DirectoryDisplay. Notifier () -> Directory .RemoveEntry ()
```

(그림 5) Directory_Display 결합 명세



(그림 6) AddEntry() 서비스에 대한 상호동작조건

이로서 결합에 참여하는 각 컴포넌트의 상호동작에 대하

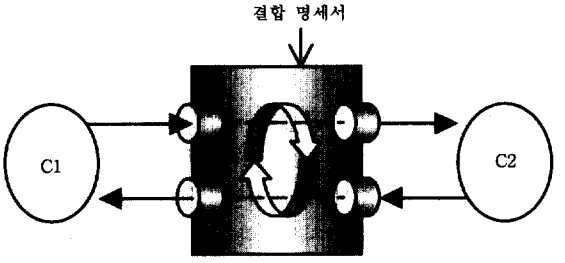
여 문맥적 의미를 명세할 수 있는 방법을 제시하고 있고 복합 컴포넌트 사용자는 복합 컴포넌트에서 제공하는 서비스인 AddEntry(), RemoveEntry()를 사용하여 다시 재귀적 복합 컴포넌트를 작성하는데 있어 보다 정확한 문맥을 이해하며 작성하기 때문에 복합 컴포넌트의 작성 방법으로 보다 유연하고 바람직하다고 본다.

3. 결합 명세서에 기반 한 결합 모델

결합 명세의 개념을 도입한 결합 모델은 재귀적 컴포넌트 결합과 다중 티어 아키텍처를 지원할 수 있으며 이들을 바탕으로 <표 2>의 컴포넌트 결합형태를 모두 지원할 수 있다.

3.1 재귀적 컴포넌트 결합의 지원

결합 명세서는 기존의 컨트랙트와는 달리 (그림 7)과 같이 컴포넌트들이 결합될 수 있도록 연결자(connector)의 역할을 수행한다. 연결자로써의 결합 명세서는 컴포넌트 C1과 C2를 연결시키면서 두 컴포넌트간에 일어나는 상호동작 패턴을 관리한다. 결합 명세서에 명세 된 상호동작 패턴의 제약 조건을 이용하면 기존의 특정 컴포넌트에 대한 컨트랙트에서는 명세할 수 없었던, 독립적으로 개발되어진 컴포넌트들간의 상호동작 패턴(interaction pattern)을 관리할 수 있다.



(그림 7) 연결자로써의 결합 명세서

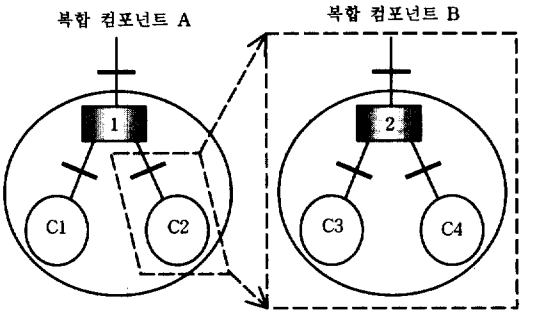
상호동작 패턴의 관리는 하나의 컴포넌트가 작성되는 시점에서 고려하지 않은 다른 컴포넌트와 결합되어서 상호동작을 할 때, 각 컴포넌트들이 제공하는 서비스들이 다른 컴포넌트들에 의해서 올바르게 사용될 수 있도록 할 수 있다.

(그림 8)과 같이 재귀적 컴포넌트 결합에 의해서 만들어진 복합 컴포넌트가 다른 결합 명세서에게 컴포넌트로써 보이기 위해서는, 먼저 결합 명세서가 연결자로써의 역할을 수행해서 컴포넌트들을 결합시킨 다음에 결합 명세서는 컴포넌트 결합 후에 생성되는 복합 컴포넌트에 대한 컨트랙트로써의 역할도 수행해야 한다. 이와 같이 되면 결합 명세서에 의해서 만들어진 복합 컴포넌트는 다시 다른 결합 명세서에 의해서 다른 컴포넌트들과 결합될 수 있다.

(그림 8)에서 컴포넌트 C3과 C4는 결합 명세서 2에 의해서 결합되어 복합 컴포넌트B가 생성된다. 복합 컴포넌트 B는 결합 명세서 2에 의해서 제공되어진 컨트랙트를 이용, 컴포넌트 C2로써 결합 명세서 1에 의해서 컴포넌트 C1과 결합되어서 복합 컴포넌트 A를 만들게 된다. 마찬가지 방법으로 복합 컴포넌트A는 복합 컴포넌트 B가 결합 명세서 1에게 일반 컴포넌트 C2로 보여질 수 있었던 것처럼 다른 결합 명세서에게 일반 컴포넌트처럼 보여질 수 있다.

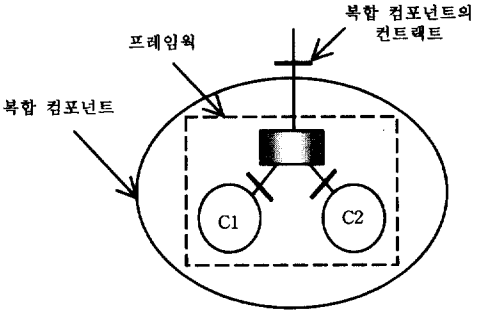
3.2 다중 티어 아키텍처의 지원

다중 티어 아키텍처의 특징은 프레임웍 자체를 컴포넌트로 볼 수 있는 것이다. 다시 말하면 하위 티어(low tier)는 상위 티어(high tier)에게 컴포넌트로 보이지만 하위 티어 자체는 프레임웍일 수도 있는 것이다. 이와 같은 특징은 결합 명세서에 의해서 재귀적으로 만들어진 복합 컴포넌트를 프레임웍으로 보면 만족시킬 수 있다.



(그림 8) 결합 명세서에 의한 재귀적 컴포넌트 결합

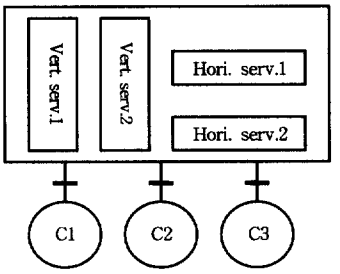
(그림 9)의 결합 명세서가 컴포넌트 C1과 C2를 결합시키면서 재귀적 컴포넌트 결합에 의한 복합 컴포넌트를 생성하고 복합 컴포넌트의 컨트랙트를 만들기 때문에 하나의 컴포넌트로써 상위 티어에 배치될 수 있다(상위 티어에게 컴포넌트로 보임). 또한 복합 컴포넌트의 내부적인 측면에서는 컴포넌트 C1, C2와 이 둘을 연결시킨 결합 명세서가 하나의 프레임웍을 구성함으로써 다중 티어 컴포넌트 아키텍처를 지원한다.



(그림 9) 결합 명세서와 프레임웍

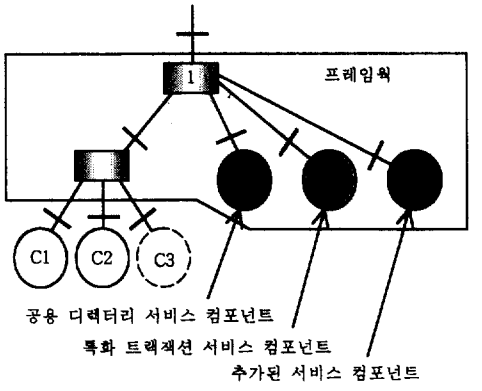
(그림 10)과 같이 현재 대부분의 프레임웍들은 프레임웍이 만들어질 때부터 그 프레임웍이 제공할 수 있는, 도메인

에 특화 된 서비스(vertical service)와 도메인 전반에 걸쳐 공통으로 사용되는 서비스(horizontal service)의 종류와 수준이 결정되며, 나중에 각 서비스의 기능 강화나 새로운 서비스의 추가가 쉽지 않다(정적 컴포넌트 프레임웍). 그러나, 기존 프레임웍에서 제공되는 도메인 특화 서비스와 도메인 공용 서비스들을 모두 컴포넌트로 만들 수 있다면 즉, 도메인 특화 서비스 컴포넌트와 도메인 공용 서비스 컴포넌트를 만들 수 있다면 기존 서비스 컴포넌트의 교체 및 새로운 서비스 컴포넌트와의 결합을 통해서 각 서비스의 기능 강화와 새로운 서비스의 추가를 쉽게 할 수 있는 동적 컴포넌트 프레임웍을 구성할 수 있을 것이다. 이와 같이 서비스 컴포넌트들을 이용해서 컴포넌트 프레임웍을 만들기 위해서는 재귀적 컴포넌트 결합과 다중 티어 아키텍처의 개념이 반드시 지원되어야 한다.



(그림 10) 정적 컴포넌트 프레임웍

결합 명세서에 의해서 만들어진 복합 컴포넌트는 앞에서 설명된 바와 같이 상위 티어에게는 컴포넌트로 보여지고, 복합 컴포넌트의 내부는 프레임웍으로 볼 수 있기 때문에 (그림 11)과 같은 프레임웍을 구성할 수 있다.

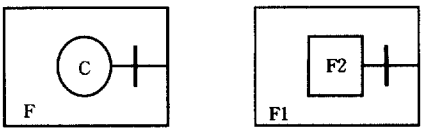


(그림 11) 동적 컴포넌트 프레임웍

(그림 11)의 프레임웍 초기상태가 도메인 공용 서비스인 디렉터리 서비스를 제공하는 컴포넌트와 도메인 특화 서비스인 트랜잭션 서비스를 제공하는 컴포넌트, 그리고 프레임웍에 배치되어서 실행되는 컴포넌트C1과 C2로 구성되어 있었다고 가정한다. 나중에 사용자의 요구에 의해서 프레임웍에 새로 추가된 컴포넌트C3 가 새로운 서비스를 요구한다면 컴포넌트C3의 요구사항을 구현한 서비스 컴포넌트(컴포넌트 내부는 요구되는 서비스가 구현된 프레임웍일 수도 있다)를 결합 명세서1의 수정을 통해서 프레임웍에 결합시킨다.

3.3 다양한 결합 형태의 지원

<표 2>의 다양한 결합형태[2]는 결합 명세서에 의한 재귀적 컴포넌트 결합과 다중 티어 아키텍처의 지원으로 프레임웍 배치 결합형태는 컴포넌트 배치 결합형태로 처리할 수 있으며 이질간의 결합, 프레임웍 확장, 그리고 컴포넌트 어셈블리는 단순 결합형태로 처리될 수 있다.



(1) 컴포넌트 배치 (2) 프레임웍 배치

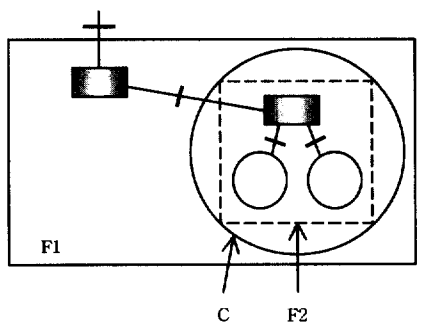
(그림 12) 컴포넌트 배치와 프레임웍 배치 결합형태[2]

프레임웍 배치 결합형태는 (그림 12)의 (2)와 같이 하나의 프레임웍이 컨트랙트를 제공함으로 해서 다른 프레임웍에 결합될 수 있는 형태를 나타낸다. 이 때 프레임웍이 제공하는 컨트랙트는 컴포넌트가 프레임웍에 배치되기 위해서 컴포넌트가 가지고 있는 컨트랙트와 같다. 이와 같이 되기 위해서는 프레임웍을 컴포넌트로 볼 수 있는 다중 티어 아키텍처의 개념이 필요하다.

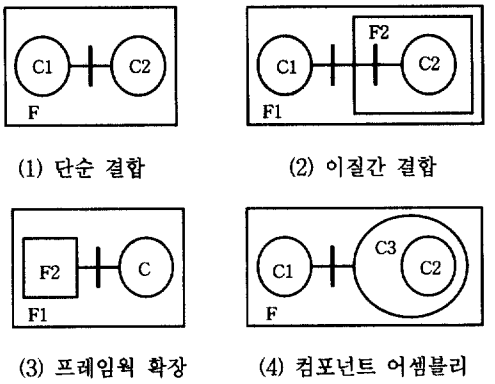
(그림 9)에서 설명된 것과 같이 결합 명세서를 이용하면 프레임웍을 컴포넌트로 볼 수 있기 때문에 (그림 12)의 (2)는 다음의 (그림 13)처럼 표현될 수 있다.

(그림 14)의 이질간의 결합, 프레임웍 확장, 그리고 컴포넌트 어셈블리 결합형태도 재귀적 컴포넌트 결합과 다중 티어 아키텍처의 개념이 지원된다면 단순 결합형태로 통합시킬 수 있다.

이질간의 결합형태에서 프레임웍 F2를 내부에 컴포넌트 C2를 포함한 복합 컴포넌트로 보면 컴포넌트 어셈블리와 같은 형태의 결합이 되며, 컴포넌트 어셈블리 결합형태는 컴포넌트 C3가 하나의 복합 컴포넌트로써 C1과 결합되는



(그림 13) 결합 명세서를 이용한 프레임웍 배치 결합형태



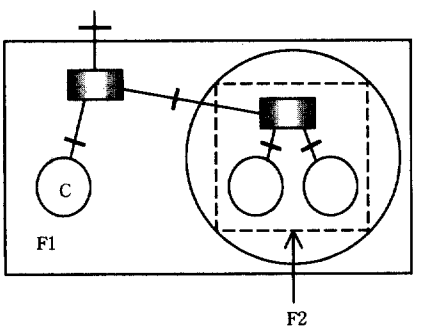
(1) 단순 결합 (2) 이질간 결합

(3) 프레임웍 확장 (4) 컴포넌트 어셈블리

(그림 14) 다양한 결합형태[2]

형태이기 때문에 단순 결합형태와 같아진다. 또한 프레임웍 확장의 형태는 프레임웍 F2가 컴포넌트로써 컴포넌트 C와 결합될 수 있기 때문에 이 역시 단순 결합과 같은 형태의 결합형태가 된다.

단순 결합과 컴포넌트 어셈블리 결합형태는 (그림 8)과 (그림 9)를 통해서 지원됨을 알 수 있다. 프레임웍 확장 결합형태는 (그림 11)의 컴포넌트 C3과 같이 기존의 결합 명세서를 수정해서 프레임웍과 컴포넌트를 결합시키는 방법이 있을 수 있으며, (그림 15)와 같이 새로운 결합 명세서를 이용해서 프레임웍 F2에 컴포넌트 C를 결합시키는 방법으로 지원할 수도 있다.



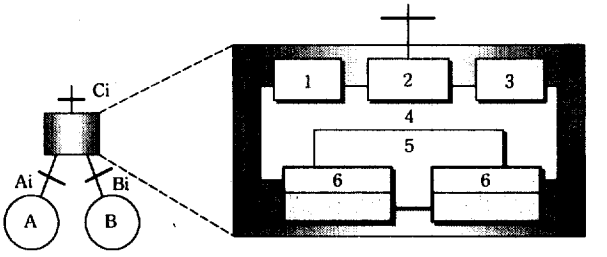
(그림 15) 결합 명세서를 이용한 프레임웍 확장 결합 형태

4. 결합 명세서를 지원하는 컴포넌트 모델의 구현 방안

결합 명세서는 (그림 7), (그림 8)과 같이 하위 컴포넌트들을 연결시켜주는 연결자로써의 역할과 재귀적 컴포넌트 결합에 의한 복합 컴포넌트의 인터페이스를 외부로 보여줄 수 있어야 한다. 이와 같은 요구 조건을 만족시키기 위해서 결합 명세서는 연결자(connector)로 구현되어질 수 있다.

4.1 연결자의 내부 구성 요소

(그림 16)은 결합 명세서를 연결자로써 구현하고자 할 때, 연결자가 가질 수 있는 구성 요소를 나타낸다. 앞으로의 설명을 위해서 결합에 참여하는 컴포넌트 A와 B는 하위 컴포넌트, 컴포넌트 A와 B의 인터페이스는 각각 Ai와 Bi, 컴포넌트 모델이 다른 하위 컴포넌트들은 이종 컴포넌트, 결합 명세서의 내용을 구현한 연결자 C와 연결자 C에 의해서 재귀적으로 만들어지는 컴포넌트는 복합 컴포넌트, 그리고 복합 컴포넌트의 인터페이스는 Ci라고 정의한다. 따라서 (그림 16)은 결합 명세서를 구현한 연결자 C가 인터페이스 Ai와 Bi를 이용, 컴포넌트 A, B를 결합시켜서 복합 컴포넌트를 만들고 복합 컴포넌트의 인터페이스인 Ci를 외부로 보여주는 상황을 나타낸 것이다.



1 : 통신 관리자(Communication Management)
2 : 인터페이스 저장소(Interface Repository)
3 : 인터페이스 해석기(Interface Translator)
4 : 라우터(Router)
5 : 서비스 관리자(Service Management)
6 : 컴포넌트 프록시(Component Proxy)

(그림 16) 연결자 내부 구조

연결자의 각 구성 요소에서 요구되어지는 기능과 역할은 다음과 같다.

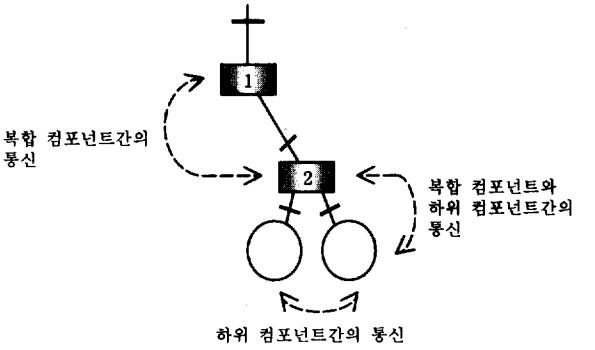
- 통신 관리자
  복합 컴포넌트간의 통신을 위한 비동기식 메시지 전송 메커니즘을 제공해야 한다.
- 인터페이스 저장소
  하위 컴포넌트의 인터페이스 Ai와 Bi뿐만 아니라 복합 컴포넌트의 인터페이스 Ci도 저장하며 복합 컴포넌트의 인터페이스 Ci를 외부로 보여줘야 하는 역할을 수행한다. 인터페이스 저장소에 의해서 나타나는 복합 컴포넌트의 인터페이스 Ci는 다른 연결자에 의해서 다시 상위 수준의 복합 컴포넌트가 만들어질 때 사용되며, 저장된 하위 컴포넌트의 인터페이스Ai와 Bi는 메시지의 라우팅 결과로 하위 컴포넌트의 서비스를 요청할 필요가 있을 때 사용된다.
- 인터페이스 해석기
  인터페이스 해석기는 하위 컴포넌트의 인터페이스 Ai, Bi 및 결합 명세서의 내용을 분석할 수 있어야 하며 결합 명세서에서 복합 컴포넌트의 인터페이스 Ci를 추출, 이를 인터페이스 저장소에 저장할 수 있어야 한다.
- 라우터
  통신 관리자가 수신한 메시지를 분석한 후, 인터페이스 해석기에 의해서 만들어진 내용을 기반으로 하위

컴포넌트의 서비스를 호출한다. 실제 서비스의 호출은 컴포넌트 프록시에 의해서 이루어지기 때문에 라우터는 어떤 하위 컴포넌트의 어떤 서비스를 호출할 것인지 결정한다.

- 서비스 관리자
  서비스 관리자는 통합적인 naming서비스와 생성 주기(lifecycle)서비스를 관리한다. Naming서비스를 예로 들면, 자바빈즈(JavaBeans) 컴포넌트와 마이크로 소프트의 COM 컴포넌트가 같은 naming서비스를 요구하지만 자바빈즈의 계층적인 이름(hierarchical naming)과 COM에서 사용되는 GUID(Global Unique ID)와 같이 naming전략이 다를 수 있기 때문에 통합적인 naming 서비스의 관리가 필요하다.
- 컴포넌트 프록시
  컴포넌트 프록시는 하위 컴포넌트가 제공하는 서비스의 호출과 이종 컴포넌트들의 인터페이스를 해석하기 위해서 필요한 구성 요소로 (그림 16)과 같이 크게 두 부분으로 나눠지며 상단부분과 하단부분은 상호배타적으로 실행된다. 하위 컴포넌트가 복합 컴포넌트인 경우에는 상단부분만이 실행되고 복합 컴포넌트가 아닐 경우에는 하단부분만이 실행된다. 상단부분은 라우터에 의해서 전달된 메시지의 추상화 수준을 조절하는 기능을 수행하며 사선으로 표시된 하단부분은 컴포넌트에 종속적인(component dependent) 부분으로 하위 컴포넌트의 모델에서 요구하는 통신 방법을 이용한 컴포넌트 서비스의 호출과 데이터 변환(marshaling/unmarshaling) 기능을 수행한다.

4.2 연결자의 동작방식

연결자의 동작방식은 컴포넌트간의 통신종류와 하위 컴포넌트가 결합되는 시점, 그리고 연결자의 실행 시점에 따라서 분류할 수 있다. 컴포넌트간의 통신종류는 (그림 17)과 같이 총 세 가지로 분류할 수 있다. 컴포넌트의 결합 시점은 다양하게 세분화할 수도 있지만 본 논문에서는 정적 결합과 동적 결합의 두 가지 형태로 분류한다. 정적 결합은 하위 컴포넌트의 구현부분에서 결정된 것으로 연결자가 컴포넌트 통신에 관여하지 않고 하위 컴포넌트간에 직접 통신이 이루어지며 동적 결합은 하위 컴포넌트의 구현부분에서 결정된 것이 아닌 (그림 1)의 요구부분에 명세된 것으로 연결자가 컴포넌트 통신에 관여하며 하위 컴포넌트들은 연결자를 통한 간접 통신을 하게 된다. 하위 컴포넌트들이 연결자를 통한 간접 통신을 하기 때문에 연결자는 상황에 따라서 하위 컴포넌트를 동적으로 교체할 수도 있다. 마지막 연결자의 실행 시점은 연결자가 복합 컴포넌트의 생성에 관여할 때와 연결자가 복합 컴포넌트의 실행에 관여할 때의 두 가지 경우로 나눈다.



(그림 17) 컴포넌트간의 통신 종류

연결자에 의해서 만들어지는 복합 컴포넌트간의 통신은 메시지를 기반으로 하며 각 복합 컴포넌트가 인터페이스를 통해서 외부로 제공하는 메시지는 그 복합 컴포넌트의 서비스 추상화 수준에 부합되는 추상화 수준을 가진다. (그림 17)의 경우에 복합 컴포넌트 1은 복합 컴포넌트 2보다 서비스 추상화 수준이 높으며 각 복합 컴포넌트에서 처리 가능한 메시지의 추상화 수준도 복합 컴포넌트 1의 메시지 추상화 수준이 복합 컴포넌트 2의 것보다 높다.

연결자와 메시지 기반의 통신 방법을 제안한 C2 아키텍쳐[11]에서는 사용자 인터페이스 컴포넌트(UI Component)들을 연결자를 이용해서 결합시킨다. C2 아키텍쳐의 연결자의 주 기능은 메시지의 라우팅이며 메시지의 전송은 비동기(asynchronous)식 브로드 캐스팅(broadcasting)을 이용한다. 비동기식 브로드 캐스팅의 메시지 전달은 컴포넌트 서비스의 함축적인 호출(implicit invocation)[12]을 가능하게 하며 분산 환경에 더욱 적합할 수 있다[11]. 그러나 비동기식 브로드 캐스팅 메시지 전달 방식은 MVC모델 등의 사용자 인터페이스 컴포넌트의 사용 패턴에 잘 부합될 수는 있지만 브로드 캐스팅 전략을 사용하기 때문에 일반적인 컴포넌트간 통신 방법에는 적합하지 않을 수도 있다. 따라서 복합 컴포넌트간의 통신은 비동기식 메시지 전송을 기반으로 하며 브로드 캐스팅은 사용하지 않는다.

연결자는 이종 컴포넌트들을 결합시켜서 복합 컴포넌트를 만들 수 있다. 하지만 이종 컴포넌트들이 사용하는 인터페이스의 내용과 문법은 서로 다르며 컴포넌트간 통신하는 방법자체도 다를 수 있기 때문에 연결자가 이종 컴포넌트들을 결합시킬 수 있도록 하기 위해서는 연결자가 이종 컴포넌트들의 인터페이스를 해석할 수 있어야 하며 또한 일관된 방법으로 하위 컴포넌트들과의 통신을 할 수 있어야 한다. CCA(Common Component Architecture)[10]는 표준화를 통한 일관된 방법을 이용하여 이종 컴포넌트들의 인터페이스를 명세할 수 있어야 하며 표준화된 통신 방법이 필요하다고 설명하고 있다. 그래서 CCA는 이종 컴포넌트들의 인터페이스를 일관된 방법으로 명세 할 수 있는 SIDL (Scientific Interface Definition Language)과 CCA 프레임웍이 이종 컴포넌트들과 일관된 방법으로 통신하기 위한

Port개념을 이용해서 이종 컴포넌트들을 자유롭게 사용할 수 있도록 하고 있다. 그러나 이종 컴포넌트들의 인터페이스를 해석할 수 있는 컴포넌트 종속적인 모듈을 이용하고 이 모듈들이 이종의 인터페이스를 연결자가 인식 가능한 형태의 인터페이스로 변환해줄 수 있으면 SIDL과 같이 일관된 인터페이스 명세방법을 이용하는 것과 동일한 결과를 기대할 수 있기 때문에 본 논문에서 제안하는 연결자 구현 방안에서는 컴포넌트 프록시가 이종 컴포넌트의 인터페이스 해석과 통신을 동시에 담당하게 된다. 복합 컴포넌트간의 통신은 통신 관리자, 복합 컴포넌트의 인터페이스는 인터페이스 해석기에 의해서 해석되며 하위 컴포넌트와의 통신과 하위 컴포넌트의 인터페이스 해석은 컴포넌트 프록시에 의해서 수행되어진다.

이종 컴포넌트간의 통신 방법은 CORBA-OLE 또는 OLE-CORBA와 같은 브리지(bridge) 명세에 대한 활발한 연구와 구현이 진행되고 있기 때문에 본 논문에서는 다루지 않는다.

지금까지 설명된 통신 종류와 이에 따른 결합 시점과 연결자 실행 여부를 조합하면 <표 1>과 같이 총 12가지의 연결자 동작방식이 나올 수 있다. 각 경우에서 연결자의 내부 구성 요소들이 서로 어떻게 동작하는지를 간단히 살펴보면 연결자에 의한 복합 컴포넌트의 생성과 연결자의 구현 가능성을 알 수 있을 것이다. 각 방식에서 실행되는 연결자 구성 요소들의 순서를 알아보기 쉽게 표현하기 위해서 "→" 기호를 사용한다.

〈표 1〉 연결자의 동작방식 분류

통신 종류	결합 시점	연결자 실행 시점	
		복합 컴포넌트의 생성	복합 컴포넌트의 실행
복합 컴포넌트-복합 컴포넌트	정적	(1)	(2)
	동적	(3)	(4)
복합 컴포넌트-하위 컴포넌트	정적	(5)	(6)
	동적	(7)	(8)
하위 컴포넌트-하위 컴포넌트	정적	(9)	(10)
	동적	(11)	(12)

(1)

인터페이스 해석기(복합 컴포넌트 인터페이스 해석)→라우터를 위한 정보 생성→인터페이스 저장소에 저장→컴포넌트 프록시 생성

(2), (4)

통신 관리자(수신된 메시지를 해석)→라우터에게 전달→라우터(인터페이스 해석기에 의해서 만들어진 정보를 이용, 컴포넌트 프록시를 선택)→컴포넌트 프록시에게 메시지 전달→컴포넌트 프록시(메시지의 추상화 수준 변경)→컴포넌트 프록시와 연결된 복합 컴포넌트에게 메시지 전송

(3)

인터페이스 해석기(연결자 인터페이스 해석)→서비스 관리자(복합 컴포넌트 인터페이스가 요구하는 서비스가 있는지 검사)→라우터를 위한 정보 생성→인터페이스 저장소에 저장→컴포넌트 프록시 생성

(5), (7), (9), (11)

컴포넌트 프록시(하위 컴포넌트 인터페이스 해석)→서비스 관리자(하위 컴포넌트에서 요구하는 서비스가 있는 검사)→라우터를 위한 정보 생성→인터페이스 저장소에 저장

(6), (8)

통신 관리자(수신된 메시지를 해석)→라우터에게 전달→라우터(컴포넌트 프록시에 의해서 만들어진 정보를 이용, 컴포넌트 프록시를 선택)→컴포넌트 프록시에게 메시지 전달→컴포넌트 프록시(메시지와 매핑 되는 컴포넌트의 메소드 선택)→하위 컴포넌트의 통신 방법에 따라 컴포넌트의 메소드 호출

(10), (12)

하위 컴포넌트에 의해서 요구된 서비스는 (9), (11)에서 이미 실행이 되었기 때문에 연결자의 구성 요소가 관여할 필요가 없다.

복합 컴포넌트 또는 하위 컴포넌트와의 결합 및 통신에 중요한 역할을 하는 컴포넌트 프록시는 (1)과 (3)의 경우에 인터페이스 해석기에 의해서 생성되지만 (5), (7), (9), (11)의 경우에는 결합 명세서, 연결자, 그리고 본 논문에서 제안하는 컴포넌트 아키텍처를 지원하는 도구에 의해서 연결자 내에 생성된다. 또한 위의 동작방식 설명에서 언급하지 않은 상호동작 제약 조건도 도구에 의해서 검증이 가능하기 때문에 연결자 자체에서는 고려하지 않는다.

5. 관련연구 및 비교

5.1 컨트랙트

컴포넌트에 대한 명세 방법은 기존의 프로그래밍 언어들에서 사용되는 API(Application Programming Interface) 방식, 객체지향 방식에서 사용되는 인터페이스 또는 CORBA 등에서 사용되는 특정 언어 중립적인 IDL(Interface Definition Language)등과 같이 자신의 기능을 정해진 문법적 규칙에 따라 명세하는 방법 등이 있지만 제 3자에 의해 독립적으로 결합에 참여하기 위하여는 많은 정보가 필요하며 이러한 정보를 컴포넌트의 문맥적 의미라 한다. 컴포넌트에 대한 인터페이스 및 상호동작에 대한 문맥적 조건까지 명세한 것을 컨트랙트적 인터페이스 또는 컨트랙트라 한다. 문맥적 정보는 크게 다음과 같이 3가지로 구분되어진다.[2]

- 행위 속성(behavioral property)
- 동기화 속성(synchronization property)
- QoS 속성(quality of service property)

이중 가장 많은 연구가 이루어지고 있는 부분은 behavioral property의 명세이며 이는 다시 전조건(precondition), 후조건(postcondition)을 이용한 명세 방법과 대수적 명세 방법(algebraic specification)에 기반을 둔 permissible history를 명세하는 방법, Refinement Calculus에 기반을 둔 abstract statement를 이용하는 방법이 제시되어 있다[8].

그러나 이러한 방법들은 모두 특정 컴포넌트의 특정 기능에 대하여 명세되는 방식으로 제안되어 있을 뿐 본 논문에서 제시하는 바와 같이 결합에 참여하는 각 컴포넌트의 각 기능의 상호동작에 대한 조건 명세 방안으로 제시된 바는 없다. 이들 방법들 중 본 논문에서 제시하는 결합 명세서와 가장 관련이 있는 것은 permissible history를 명세하는 방법으로 본 논문에서 제시하는 결합 명세서에서는 상호동작 조건을 표현하기 위한 방법으로 이 명세 방법을 사용하였다.

동기화 속성에 대해서는 object calculus[16], piccola[14] 등에서 Hoare's CSP[17]나 Milner's pi-calculus[18]등을 이용한 명세 방안을 제시한 바 있으나 본 논문에서 제시하는 결합 명세서의 명세 방안에는 고려되어 있지 않으며 추후 보완할 예정이다.

컴포넌트간 결합에 대한 명세 방법에 대하여 많은 연구가 진행되고 있다. 그 중 스크립트 언어를 이용하여 컴포넌트를 결합하려는 노력이 있다. piccola[14]는 언어의 디자인 요소를 컴포넌트와 컴포넌트 결합을 명세하는 script와 컴포넌트의 인터페이스나 컨트랙트를 적용시킬 때 필요한 glue로 구분하여 프로그램 로직과 결합에 필요한 요소를 구분하고 있지만 재귀적 결합을 지원하고 있지 않다.

또한 Rapide, Darwin, Aseop, Unicon, Wright, ACME등 ADL(Architecture Description Language)에서는 컴포넌트를 결합하기 위하여 결합에 참여하는 컴포넌트를 연결 시켜주는 connector를 정의하고 있다[15]. 각 컴포넌트는 port, player, constituent 등으로 불리는 제공과 요구 부분으로 구분되어지는 인터페이스를 가지며 이들을 외부적으로 구성하여 연결하거나 언어에 인라인(in-line)으로 한 컴포넌트의 요구와 다른 컴포넌트의 제공을 바인딩하여 연결하는 connector를 이용하여 컴포넌트를 결합한다. Connector는 connector type에 맞는 컴포넌트를 연결해주며 연결되어 있는 컴포넌트간 상호동작에 대한 조건 명세를 가지고 있지 않다.

컴포넌트의 재귀적 결합에 대해서 언어적 측면에서 접근한 방법[1]이 제안되었다. 이 연구에서는 XML기반의 BML (Bean Markup Language)을 제안하고 이를 이용해서 자바빈즈(JavaBeans) 컴포넌트의 재귀적 결합과 이들 사이의 자바 이벤트 바인딩은 비교적 잘 명세되고 있으나 결합에 참여하는 자바빈즈 컴포넌트들간의 기능적 상호동작에 관한 제약조건의 명세방법에 대해서는 충분히 고려되지 않았다.

본 논문에서 제안하는 결합 명세서의 명세 방안은 결합에 참여하는 컴포넌트간 상호동작에 대한 조건을 명세하여 상호동작에 대한 명세 방법을 제안하고 있으며, 결합에 참여하는 컴포넌트들의 명세 부분을 그대로 수용하고 이를 다시 복합 컴포넌트의 제공과 요구 서비스로 분류하여 복합 컴포넌트의 명세 부분으로 이용함으로써 재귀적 복합 컴포넌트를 명세하는 방법을 제안하고 있다.

5.2 다양한 컴포넌트 결합 형태

컴포넌트 기반 시스템에서 결합의 요소로는 컴포넌트와 프레임웍을 들 수 있으며 이들 요소로 결합의 형태를 크게 세 가지로 구분할 수 있는데 컴포넌트간 결합, 컴포넌트와 프레임웍의 결합, 프레임웍간의 결합으로 들 수 있다. 이들을 다시 세분하여 결합의 형태를 다음과 같이 6가지로 구분하고 있으며[2] 각각의 경우에 따라 명세되어지는 컨트랙트의 내용도 구분하고 있다.

- 컴포넌트 배치 : 컴포넌트가 프레임웍에 배치
- 프레임웍 배치 : 프레임웍이 다른 프레임웍에 배치
- 단순 결합 : 같은 프레임웍에 배치되어있는 컴포넌트간 결합
- 이질간 결합 : 다른 프레임웍에 배치되어있는 컴포넌트간 결합
- 프레임웍 확장 : 같은 프레임웍에 배치되어있는 컴포넌트와 프레임웍의 결합
- 포넌트 어셈블리 : 같은 프레임웍에 배치되어있는 컴포넌트와 복합 컴포넌트간 결합

이들 각 결합 형태에 대해 현재 주 컴포넌트 모델에서의 비교가 <표 2>[2]와 같이 제시되고 있다.

각 컴포넌트 모델은 컴포넌트 배치에 대한 배치 컨트랙트는 정의하고 있다. 그러나 그 외의 다른 컴포넌트 결합

〈표 2〉 각 컴포넌트 모델에서의 결합 형태

Form/ Technology	EJB	COM+	Java Bean	Water Bean	OMG/Orbos
Component Deployment					
Framework Deployment	Future (container contract)		(JVM plug-in)		(Portable object adapter)
Simple Composition					
Heterogeneous Composition	(IIOP)				(IIOP)
Framework Extension		Future (policy objects)			
Component Assembly					

형태에 대한 명세서의 정형적인 명세 구조에 대해서는 아직 일치하고 있지 않다. 또한 단순 결합과 컴포넌트 어셈블리에 대한 명세 구조에 대해서는 아직 연구가 충분히 진행되고 있지 않다. 본 논문에서는 이들 다양한 결합 형태를 지원하기 위한 결합 명세의 명세 구조를 제안하고 있으며 또한 각 결합 형태에 대하여 통일된 결합 명세서 명세 구조를 제안하고 있다.

6. 결론 및 향후 계획

기존의 컴포넌트 명세 방법들은 특정 컴포넌트에 대한 기능적 명세에 상호결합을 위한 문맥 정보를 추가하는 형식을 취함으로써 상호결합을 위한 명세서로서의 기능에 한계를 가질 뿐만 아니라 컴포넌트의 재귀적 결합에 대한 지원 기능도 불충분하다.

이에 대한 보완책으로 본 논문에서는 결합에 참여하는 컴포넌트들의 명세와 구분하여 별도의 결합 명세서라는 명세 구조를 제안하였으며, 이를 이용하여 컴포넌트의 다양한 결합 형태의 지원에 대하여 연구하였다.

결합 명세서는 기존 컴포넌트 명세 방법으로는 표현하기 어려운 복합 컴포넌트간의 상호동작 조건의 명세를 가능하게 해주며, 재귀적 컴포넌트 결합을 통하여 다양한 형태의 컴포넌트 결합을 일관된 형식으로 지원한다. 이는 기존의 컴포넌트 아키텍처에 비하여 컴포넌트의 크기 및 컴포넌트 모델에 대하여 좀 더 유연성(flexibility)과 확장성(scalability)이 높은 컴포넌트 아키텍처의 구축을 가능하게 할 뿐만 아니라 다중 티어 아키텍처의 구축도 가능하게 한다. 또한 결합 명세서는 컴포넌트 결합을 위한 연결자로 구현가능 하므로 컴포넌트 결합 도구의 구축에도 활용될 수 있을 것으로 판단된다.

앞으로 결합 명세서에 동기화 속성(synchronization properties)과 서비스의 질(QoS) 부분의 추가 방법, 상호동작 조건의 사전 검증 방법에 대하여 계속 연구를 진행할 예정이며 자바 환경에서 결합 명세서를 이용한 컴포넌트 결합 도구의 구현 연구를 진행하고 있다.

참 고 문 헌

[1] Francisco Curbera, Sanfiva Weerawarana, Matthew J. Duftler, "On Component Composition Language," IBM T. J. Watson Research Center, 2000.

[2] Felix Bachman, len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, Kurt Wallnau, "Volume II : Technical Concepts of Component-Based Software Engineering," CMU/SEI-2000-RT-008, 2000.

[3] A. Beugnard, J-M Jezequel, and D.Watkins. "Making Components Contract Aware," IEEE Computer, July, 1999.

[4] Clemens Szypaski, "Component Software Beyond Object-oriented Programming," Addison Wesley, 1998.

[5] Koen De Hondt, Carine Lucas, Patrick Steyaert, "Reuse Contracts as Component Interface Descriptions," WCOP 1997.

[6] Johannes Leon Marais, "Design and Implementation of a Component-Architecture for Oberon," thesis, 1996.

[7] Sametinger J., "Software Engineering with Reusable Components," Springer-Verlag, Town., 1997.

[8] 백경원, 이정태, 류기열, "상호작용 제약 조건을 기술할 수 있는 컴포넌트 결합 컨트랙트", 제16회 한국정보처리학회 추계학술발표대회, 2001.

[9] 박성은, 이정태, 류기열, "결합 컨트랙트를 지원하는 컴포넌트 시스템 아키텍처", 제16회 한국정보처리학회 추계학술발표대회, 2001.

[10] Armstrong, R. C., D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High-Performance Parallel Computing," HPDC '99, Redondo Beach, CA, pp.4-6 Aug. 1999.

[11] Richard N. Taylor, Nenad Medvidovic, Kenneth M. Anderson, E. James Whitehead Jr. and Jason E. Robbins., "A Component-and Message-Based Architectural Style for GUI Software," In Proceedings of the Seventeenth International Conference on Software Engineering (ICSE17), Seattle WA, 24-28, pp.295-304, April, 1995.

[12] Kevin J. Sullivan and David Notkin, "Reconciling environment integration and software evolution," ACM Trans. on Software Engineering and Methodology, 1(3) : pp.229-268, July, 1992.

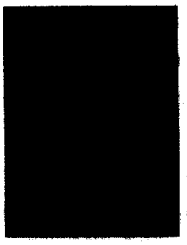
[13] Jean-Guy Scheider and Oscar Nierstrasz, "Components, Scripts and Glue," In L. Barroca, J. Hall, and P. Hall, editors, Software Architectures --Advances and Applications, pp.13-25, Springer, 1999.

[14] Ariel D. Fuxman "A Survey of Archiecture Description Languages," February, 2000.

[15] K. Lano, J. Bicarregui, T. Maibaum, J. Fiadeiro, "Composition of Reactive System Components," Proceedings of the first Workshop on Component-Based Systems, 1997.

[16] R. Allen, R. Douence, D. Garlan, "Specifying Dynamism in Software Architectures," Proceedings of the first Workshop on Component-Based Systems, 1997.

[17] C. Canal, E. Pimentel, J. Troya, "On the Composition and Extension of Software Components," Proceedings of the first Workshop on Component-Based Systems, 1997.



백 경 원

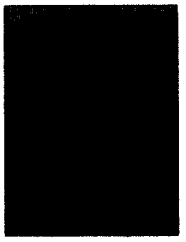
e-mail : beeback@madang.ajou.ac.kr

1996년 아주대학교 컴퓨터공학과 졸업 (공학사)

1999년 아주대학교 대학원 컴퓨터공학과 졸업(공학석사)

2001년 아주대학교 대학원 컴퓨터공학과 수료(박사수료)

연구분야 : S/W 컴포넌트, 분산 컴포넌트, 소프트웨어 공학



박 성 은

e-mail : aceoface@madang.ajou.ac.kr

1998년 아주대학교 컴퓨터공학과 졸업 (공학사)

2000년 아주대학교 일반대학원 컴퓨터공학과(공학석사)

2000년~현재 아주대학교 일반대학원 컴퓨터공학과(박사과정)

관심분야 : 컴포넌트 아키텍처, 컴포넌트 결합

이 정 태

e-mail : jungtae@madang.ajou.ac.kr

1979년 서울대학교 농과대학 학사

1981년 서울대학교 자연과학대학 계산학 이학석사

1988년 서울대학교 자연과학대학 계산학 이학박사

1982년~1983년 울산공과대학교 전산학과 전임강사

1983년~1988년 아주대학교 공과대학 전자계산학과 전임강사

1988년~1997년 아주대학교 정보통신대학 조교수

1997년~현재 아주대학교 정보통신대학 부교수

관심분야 : 객체지향시스템, 프로그래밍 언어, 컴포넌트 기술 등

류 기 열

e-mail : kryu@madang.ajou.ac.kr

1985년 서울대학교 컴퓨터공학 학사

1987년 한국과학기술원 전산학 석사

1992년 한국과학기술원 전산학 박사

1992년~1993년 한국과학기술연구원 연구원

1993년~1994년 동경대학 객원 연구원

1994년~현재 아주대학교 정보 및 컴퓨터공학부 부교수

관심분야 : 객체지향시스템, 프로그래밍 언어, 컴포넌트 기술, 메시징 시스템 등