독립적으로 개발된 바이너리 컴포넌트들의 조립을 지원하는 컴포넌트 모델

임윤선°[1], 김 명[1], 정안모[2]
이화여자대학교 컴퓨터학과[1], 리버넥스[2]
lys96@ewhain.net, mkim@ewha.ac.kr, amjeong@libernex.com

A Component Model Supporting the Assembly of Independently Developed Binary Components

Yoonsun Lim°[1], Myung Kim[1], Anmo Jeong[2]
Dept. of Computer Science and Engineering, Ewha Womans University[1]
LiberNex, Inc. [2]

요 약

컴포넌트 기반 개발 방법론이 지향하는 목표는 독립적으로 개발된 컴포넌트들을 소스 수정 없이 바이너리 형태로 조립하여 소프트웨어를 개발하는 것이다. 그러나 기존 컴포넌트 모델 기반으로 개발된 컴포넌트들은 자신이 제공하는 서비스에 대해서는 그 사용계약을 독자적인 인터페이스를 통하여 정의하는 반면, 하위 컴포넌트와의 연동은 하위 컴포넌트가 정의한 인터페이스에 따르는 코딩을 통해 이루어진다. 이러한 컴포넌트 모델들은 다계층 구조로 개발되고 있는 현대 엔터프라이즈 정보시스템에서 상위 컴포넌트들이 하위 컴포넌트가 정의한 인터페이스에 정적으로 단단히 결합되는 문제점을 갖고 있다. 따라서 하위 컴포넌트에 의존하지 않는 간단한 GUI용 컴포넌트들만 재사용되고 비즈니스 논리를 처리하는 중간 계층 컴포넌트들의 재사용율은 지극히 낮은 실정이다. 본 논문은 하위 컴포넌트가 정의한 인터페이스 규격에 따르는 대신 독자적으로 호출규격을 정의하여 사용하고 이에 대한 메타데이터를 공개함으로써 독립적으로 개발된 바이너리 컴포넌트들간의 연동을 보장하는 새로운 컴포넌트 모델을 제안하고 이 모델에 따라 개발된 컴포넌트들을 조립하는 방법을 제안하였다. 또한 이들을 구현함으로써 이미 개발된 컴포넌트를 실제 소스 수정 없이 바이너리 코드 형태로 조립 가능한 것을 증명하였다.

1. 서 론

현대 경영이 요구하는 정보 시스템은 그 복잡도와 규모가 점점 커지고 있으며, 경쟁이 심화되면서 시스템의 신속한 구축과 변경 및 확장의 용이성, 구축비용 절감 등의 필요성이 더욱 절실해졌다. 이에 대한 최적의 대안으로서 등장한 컴포넌트 기반 소프트웨어 개발 방법론(Component Based Development Methodology: CBD)은 분할 정복의 원리와 객체 지향 개발의 이점을 계승하되 소프트웨어 모듈을 부품화함으로써 재사용을 가능하게 하고 소프트웨어 개발을 조립 개념으로 발전시켰다[1,2].

컴포넌트 기반 소프트웨어 개발 방법론의 목표에 따라 컴포넌트들의 재사용성을 높이기 위해서는 컴포넌트의 소스코드를 수정하지 않고 바이너리 코드 차원에서 조립이 가능해야 한다. 그러나 기존 컴포넌트 모델에 따라 개발된 컴포넌트들은 자신이 제공하는 서비스에 대한 인터페이스는 독자적으로 정의하여 외부로 노출 시키지만, 하위 계층 컴포넌트와의 연동은 철저히 하위 컴포넌트가 정의한 인터페이스에 따른 코딩을 통해 구현된다. 이와 같은 컴포넌트 구조는 독립적으로 개발된 컴포넌트들 사이에 발생하는 인터페이스 불일치 문제를 해결하기 어렵고, 같은 인터페이스를 공유한다 해도 하위 컴포넌트 식별자가 하드 코딩 되어 있어 새로운 버전의 컴포넌트로 대체하기 어렵다는 문제점을 갖고 있다. 따라서 하위 컴포넌트 서비스에 의존하지 않는 간단한 GUI용 컴포넌트들만 재사용되고 있으며 비즈니스 논리를 처리하는 중간 계층 컴포넌트들의 재사용율은 지극히 낮은 실정이다.

본 논문은 이러한 문제점을 해결하기 위해 하위 컴포넌트가 정의한 인터페이스 규격에 따르는 대신 하위 컴포넌트 호출규격을 독자적으로 정의하여 사용하고 이에 대한 메타데이터를 외부로 노출 시키는 새로운 컴포넌트 구조를 제안하였다. 그리고 독립적으로 개발된 컴포넌트간에 필연적으로 발생하는 인터페이스

불일치 문제를 극복하기 위해 중재 컴포넌트를 통하여 조립하는 방법을 제안하였다. 또한 이 방법을 구현하는 조립 도구를 개발하여 컴포넌트들 사이의 메시지 흐름을 컴포넌트 소스 코드 수정 없이 유연하게 조절할 수 있도록 함으로써 독립적으로 개발된 바이너리 컴포넌트의 재사용을 보장할 수 있게 되었다.

본 논문은 다음과 같이 구성된다. 2장에서 하위 컴포넌트가 정의한 인터페이스를 통해 정적으로 결합된 컴포넌트들을 재사용하려 할 때 발생하는 문제점을 상세히 분석하고 3장에서 이러한 문제점을 해결하기 위한 기존 연구에 대해 소개한다. 4장에서 본 논문에서 제안한 컴포넌트 구조에 대해 설명하고, 5장에서 본 논문에서 제안한 모델에 따른 컴포넌트들을 조립하는 기술에 대해 소개한 후 6장에서 결론을 맺는다.

2. 컴포넌트 조립 문제점 분석

대규모 기업용 소프트웨어들은 대부분 다계층 구조의 분산 시스템으로 개발된다. 일반적으로 계층적 설계에서는 최상위층에 프리젠테이션 계층, 중간층에 비즈니스 로직 계층, 종단에 데이터서비스 계층을 배치한다[3]. 각 계층 사이에는 서비스를 제공하는 서버 컴포넌트와 서비스를 요청하는 클라이언트 컴포넌트 관계가 형성된다.

이러한 다계층 구조에서 종단의 컴포넌트들을 제외한 대부분의 컴포넌트들은 하위 계층의 서버 컴포넌트들과 결합되어 있으며, 하위 계층 컴포넌트가 제공하는 서비스를 바탕으로 자신의 서비스를 상위 계층의 컴포넌트에게 제공한다. 이때, 하위(서버) 컴포넌트에 대한 서비스 호출은 정적으로 코딩된 채 컴파일되어 블랙박스 형태로 배포된다. 이렇게 call stack 방식으로 결합된 바이너리 컴포넌트를 독립적으로 재사용하여 새로운 소프트웨어로 조립하고자 할 때 다음 두 가지 큰 문제점을 피할 수 없다.

(1) 인터페이스 불일치 문제

클라이언트 컴포넌트와 서버 컴포넌트가 연동되려면 호출/피호출 메소드 시그너처가 서로 정확하게 정합되어야 한다. 그러나 서로 독립적으로 개발된 컴포넌트들의 호출/피호출 인터페이스가 일치하기는 매우 드문 일이다. 따라서 이들을 연동시키려면 서버 컴포넌트가 제공하는 서비스 인터페이스뿐만 아니라 클라이언트 컴포넌트의 호출규격에 대해서도 상세하고 정확하게 알아야 한다. 그러나 현재 산업상 표준 컴포넌트 모델에 따라 개발된 바이너리 컴포넌트들은 자신이 제공하는 서비스에 대한 인터페이스에 대한 상세한 메타데이터를 외부에서 접근할 수 있게 하는 반면, 하위 계층 컴포넌트 호출에 대한 상세한 정보는 제공하지 않는다.

(2) 서버 컴포넌트 식별자 문제

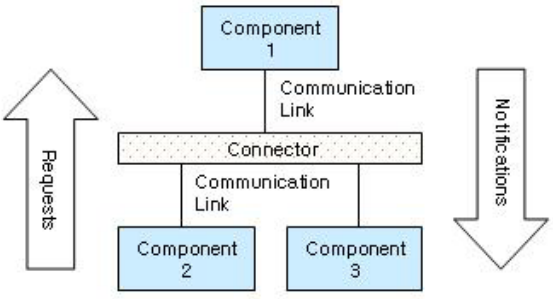
위에서 언급한 클라이언트 컴포넌트와 서버 컴포넌트간 인터페이스 정합 문제가 해결되었다 해도 클라이언트 컴포넌트에 서버 컴포넌트의 식별자가 하드코딩 되어 있어 조립을 어렵게 한다. 즉 개발 당시의 서버 컴포넌트를 나중에 독립적으로 개발된 서버 컴포넌트로 대치하는 경우 클라이언트 컴포넌트의 코드를 새로운 서버 컴포넌트 식별자로 바꿔야 하므로 컴포넌트 소스 수정 및 재 컴파일이 불가피하게 된다.

3. 기존 연구

2장에서 기술한 call stack 기반의 컴포넌트 결합에 따른 재사용의 어려움을 극복하기 위해 모든 컴포넌트들을 자율적으로 동작하게 설계하고, 이들을 워크플로우 방식으로 조립하는 방안이 연구되었다. 본 장에서는 이와 같은 워크플로우 로직을 wrapper 코드로 구현하여 컴포넌트를 조립하는 미국 C2 Style과 이에 기반한 한국전자통신연구원의 Cobalt 조립 방법에 대해 설명한다.

(1) C2 Style

C2 Style은 미국 어바인 소재의 캘리포니아 대학에서 컴포넌트 재사용을 용이하게 하기 위해 제안한 메시지 기반의 컴포넌트 조립 기술이다[4]. C2는 그림 1과 같이 컴포넌트들이 직접 통신을 하도록 하지 않고, 각 컴포넌트 사이에 connector를 두어, connector가 메시지 전송을 맡도록 한다. 또한 각 컴포넌트에는 wrapper 클래스를 만들어 메시지 전달에 필요한 상단 인터페이스(top interface)와 하단 인터페이스(bottom interface)를 지원한다. 상단 인터페이스는 컴포넌트가 상위 컴포넌트와 통신을 하는데 쓰이고, 하단 인터페이스는 하위 컴포넌트와 통신을 하는데 쓰인다. 이 인터페이스들은 connector와 연결되어 있다. 컴포넌트들의 상호연결 관계는 ADL(Architecture Description Language)을 통해 정의된다. 컴포넌트가 다른 컴포넌트로 대치되는 경우는 컴포넌트의 소스 코드 변경 없이 ADL만 조절하면 된다는 장점이 있다.



[그림1] C2 조립방법

그러나 C2는 컴포넌트 조립에 있어서 다음과 같은 문제점들을 내포하고 있다.

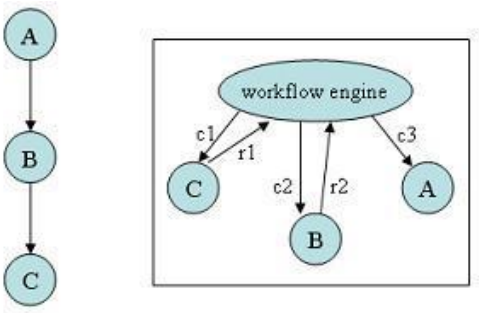
첫째, C2는 다른 메시지 기반 아키텍쳐와 마찬가지로 비동기 호출을 한다. 일반적으로 컴포넌트는 다른 컴포넌트에 서비스를 요청하고 그 서비스에 대한 결과를 요구하게 된다. 그러나 비동기 메시지 전달 방식을 취한 C2의 경우 요청한 서비스의 결과를 확인하기 위해 서비스 요청할 때 메시지를 식별할 수 있는 식별자를 부여하고 난 후 컴포넌트에게 전달되는 메시지들의 식별자를 확인하면서 요청한 서비스의 결과 메시지인지 찾아야 하는 오버헤드가 있다.

둘째, 서비스를 요청하고 요청 받는 두 개의 컴포넌트는 서버와 클라이언트의 관계이다. 일반적으로 서버 컴포넌트는 어떤 서비스를 제공할 것인지에 대한 정보를 외부에 노출하고 있다. 그러나 클라이언트 컴포넌트는 자신이 요청한 서비스가 무엇인지에 대한 정보는 외부로 노출하지 않는다. 따라서 제3자가 개발한 컴포넌트를 구입하여 조립할 때 서버 컴포넌트의 서비스뿐만 아니라 클라이언트에서 필요한 서버 컴포넌트 인터페이스를 알아야 조립이 가능하다. C2 Style 구조는 이에 대한 직접적인 해결책을 제시하지 못하는 대신 wrapper 클래스에 메시지 흐름 순서를 조정하는 코드와 하위 컴포넌트 서비스에 call stack 방식으로 의존하는 대신 필요한 모든 정보를 매개변수로 받도록 재설계된 자율적(autonomous) 컴포넌트들을 필요로 한다.

셋째, wrapper 클래스를 이용해 connector와 연결하므로 초기 설계와 다른 인터페이스를 가진 컴포넌트로 대체한 경우에는 wrapper 클래스 중재 코드에서 인터페이스를 맞춘 후 다시 컴파일해서 배포해야 하는 오버헤드가 있다. 위와 같은 문제점들은 제 3자가 개발하여 바이너리 형태로 제공되는 컴포넌트들을 조립하여 소프트웨어를 제작하는데 한계를 드러낸다.

(2) Cobalt

Cobalt는 C2에 기반한 조립 기법으로 이는 한국의 전자통신연구원에서 개발한 조립 기법이다[5]. 이는 독립적으로 개발된 컴포넌트를 조립하여 보다 큰(coarse-grained) 합성 컴포넌트(composite component)를 만들 수 있게 한다. 예를 들어, 그림 3(a)에서와 같이 컴포넌트 A가 컴포넌트 B를 호출하고 컴포넌트 B는 다시 컴포넌트 C를 호출하는 경우, Cobalt에서는 호출관계가 있는 컴포넌트 그룹마다 합성 컴포넌트를 만들고, C를 가장 먼저 호출하여 그 결과 값을 B에게 매개변수로 제공하고, B 호출의 결과 값을 C에게 매개변수로 제공하는 워크플로우 역할을 수행하는 wrapper 코드를 작성해야 한다.



(a) 표준 컴포넌트 (b) Cobalt의 composite 컴포넌트

[그림 2] Cobalt 조립 기법

Cobalt 기법은 표준 방식으로 작성된 컴포넌트들을 수용하지 못하고 하위 컴포넌트 서비스에 의존하는 대신 필요한 정보를 모두 매개 변수로 받아 자율적으로 처리하도록 만들어진 컴포넌트들만 조립할 수 있다.

Cobalt는 기존의 자연스러운 컴포넌트 계층구조를 무시한 조립도구로써 실제 개발환경에서 대형 소프트웨어를 개발하는 데 적절한 도구라고 할 수 없다. Cobalt가 갖는 또 하나의 단점은, 하나의 컴포넌트가 여러 다른 컴포넌트들과 호출관계를 갖는 경우 각 경우마다 새로운 합성 컴포넌트가 존재해야 하며, 이에 따라 설계가 부자연스럽고 매우 복잡해진다는 점이다.

(3) Dependency Injection

위에서 설명한 C2 Style과 Cobalt 기법은 인터페이스 시그너처 정합 문제에 초점을 맞춘 연구이다. 2장에서 설명한 컴포넌트 식별자 문제를 해결한 기술은 Dependency Injection 기술이다[6].

이 기술은 클라이언트 컴포넌트가 서버 컴포넌트를 직접 인스턴스화 하는 대신, 컴포넌트 컨테이너가 서버 컴포넌트를 인스턴스화 하고 그 레퍼런스를 클라이언트 컴포넌트의 속성이나 생성자 매개변수로 주입한다. 클라이언트 컴포넌트가 서버 컴포넌트 식별자를 가질 필요가 없어짐에 따라 컴포넌트 조립이 유연해진다. 하지만 Dependency Injection 기술이 컴포넌트 식별자 문제를 크게 완화시켰음에도 불구하고 클라이언트 컴포넌트와 서버 컴포넌트간 호출/피호출 인터페이스가 같아야만 하는 문제를 해결하지는 못한다. 따라서 독립적으로 개발되어 인터페이스가 상이한 바이너리 컴포넌트들은 Dependency Injection 기술로 조립할 수 없다.

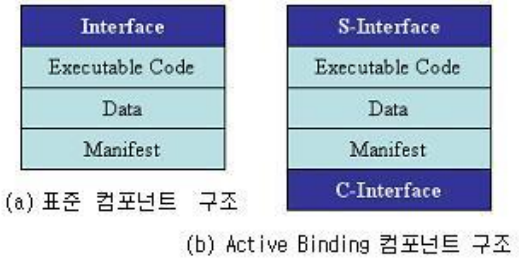
4. Active Binding 컴포넌트

2장에서 설명한 컴포넌트 조립 문제 중 첫번째 인터페이스 불일치의 문제점을 해결하기 위해 본 논문에서는 새로운 컴포넌트 구조를 제안한다. 본 논문에서 제안하는 새로운 컴포넌트를 Active Binding 컴포넌트라 부르기로 한다. 컴포넌트 조립의 두번째 문제인 컴포넌트 식별자 문제는 3장에서 기술한 Dependency Injection 기술을 Active Binding 컴포넌트 구조에 도입하여 해결한다.

(1) Active Binding 컴포넌트 구조

그림 3(a)는 닷넷 컴포넌트 구조이다. 컴포넌트는 인터페이스, 실행코드, 데이터, Manifest와 메타데이터로 구성된다. 이 중에서 컴포넌트간의 연동을 위한 것이 인터페이스이며, 메타데이터에는 컴포넌트 인터페이스 식별자와 메소드들의 시그너처(메소드 이름, 매개변수 정보, 리턴 타입 정보 등이 포함)가 포함되어 있어, 자신을 다른 컴포넌트들이 사용할 수 있도록 외부에 사용계약 정보를 제공한다.

새로 제안하는 Active Binding 컴포넌트 구조는 그림 3(b)와 같다. Active Binding 컴포넌트는 표준 컴포넌트의 인터페이스를 서버측 인터페이스(S-Interface)라고 부르기로 한다. Active Binding 컴포넌트는 표준 컴포넌트와 달리 하위(서버) 컴포넌트의 인터페이스 호출 규격에 따르는 코드를 사용하는 대신, 독자적으로 호출 인터페이스를 정의하여 이 인터페이스를 사용하여 하위 컴포넌트의 서비스를 호출하도록 코딩한다. 이 클라이언트측 인터페이스(C-Interface) 또한 서버측 인터페이스와 동일하게 메타데이터를 외부에 노출하는 구조를 갖게 함으로써 조립도구가 연결할 양측 컴포넌트의 인터페이스에 대한 상세한 정보를 동적으로 읽는 것이 가능하다.

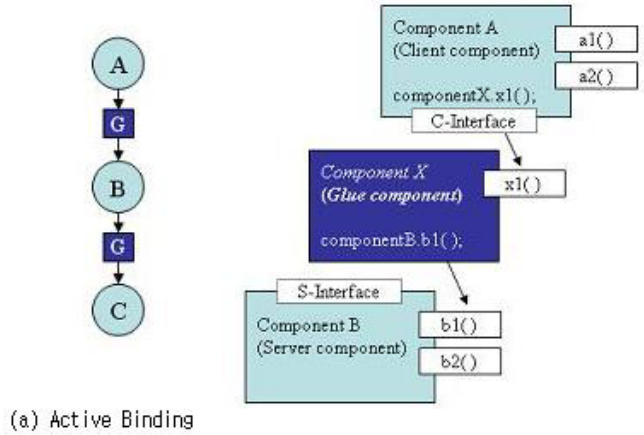


[그림 3] 표준 컴포넌트와 Active Binding 컴포넌트 구조

(2) Active Binding 컴포넌트 조립

클라이언트측 인터페이스를 갖는 Active Binding 컴포넌트를 조립하는 방법은 다음과 같다. 그림 4(a)와 같이 Active Binding 컴포넌트 A, B, C를 조립할 때 발생하는 인터페이스 불일치를 중재하기 위해, 글루 컴포넌트 G를 생성하여 컴포넌트들을 연결한다. 조립도구는 클라이언트 컴포넌트의 클라이언트측 인터페이스 메타데이터를 읽어 글루 컴포넌트의 서버측 인터페이스로 생성한다. 다음 단계로 조립도구는 서버 컴포넌트의 서버측 인터페이스를 읽어 이를 호출하는 코드를 생성한다.

좀 더 상세히 설명하면, 그림 4(b)에서 클라이언트 컴포넌트 A가 서버 컴포넌트 B의 인터페이스 b1()을 통해 제공되는 서비스를 호출해야 하는 경우, 클라이언트 컴포넌트 A가 서버 컴포넌트 B의 사용 계약 정보를 모르더라도 독자적으로 임의의 호출규격을 정의하여 사용하고, 이에 대한 상세한 정보를 메타데이터로 구축하고 이를 컴포넌트에 포함시킴으로써 조립도구가 이 메타데이터를 이용하여 클라이언트 컴포넌트에 정의된 호출규격을 지원하는 인터페이스를 구현한 글루 컴포넌트를 만들 수 있다. 또한 글루 컴포넌트 안에서 서버 컴포넌트 B의 인터페이스 b1()를 호출하여 결과 값을 클라이언트 컴포넌트 A로 전달하게 함으로써 서버 컴포넌트 B의 서비스를 클라이언트 컴포넌트 A가 호출하여 사용할 수 있다.



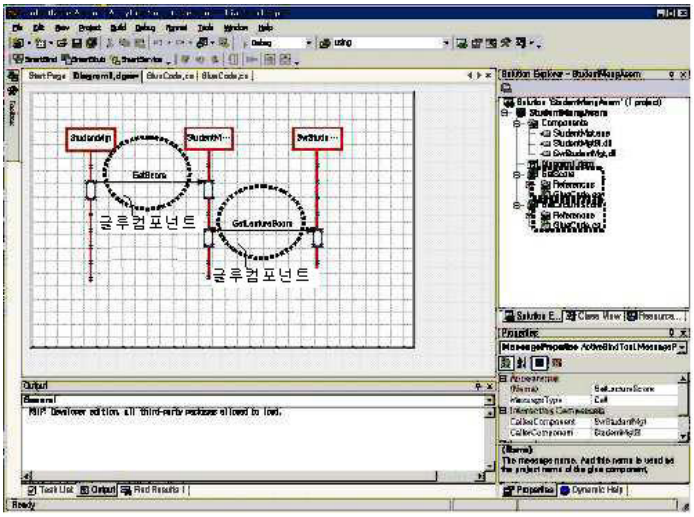
[그림 4] Active Binding 컴포넌트 조립

글루 컴포넌트가 두 컴포넌트간 구문적 불일치를 중재하는 것 외에도 컴포넌트 사이의 매개변수들 사이에 존재할 수 있는 의미론적 부정합을 중재 코드를 삽입하여 해결할 수 있다. 예를 들어 클라이언트 컴포넌트에서 서버 컴포넌트에 화폐 단위 원을 매개변수로 전달하는데, 실제 서버 컴포넌트는 달러를 매개변수로 받을 경우 글루 컴포넌트에 원을 달러로 변환하는 코드를 삽입하여 이를 중재할 수 있다.

5. Active Binding 컴포넌트 조립도구

본 논문에서 제안한 Active Binding 컴포넌트의 조립을 지원하는 조립도구 ArtComposer를 개발하였다. 그림 5에서와 같이 ArtComposer는 컴포넌트들을 시퀀스 다이어그램으로 형태로 조립할 수 있는 그래픽

사용자 인터페이스를 지원한다. 소프트웨어 개발자가 필요한 컴포넌트들을 배치하고 컴포넌트들간에 메시지 흐름을 설정하게 되면 두 컴포넌트를 연결하는 글루 컴포넌트가 자동 생성되고 개발자는 필요에 따라 중재 코드를 삽입할 수 있다.



[그림 5] Active Binding 컴포넌트 조립 도구 : ArtComposer

6. 결론

본 논문은 독립적으로 개발된 바이너리 컴포넌트들을 계층적으로 조립할 수 있는 Active Binding 컴포넌트 모델을 제안하였다. Active Binding 컴포넌트 모델은 하위 컴포넌트에 대한 호출규격을 독자적으로 정의하여 사용하는 방법을 통해 컴포넌트간 의존성을 완전히 제거함으로써 서로 다른 개발자들에 의해 독립적으로 개발된 바이너리 코드 형태의 컴포넌트들을 소스 코드 수정 없이 재사용할 수 있게 한다.

또한 서로 이질적인 호출/피호출 인터페이스를 가진 Active Binding 컴포넌트들을 서버와 클라이언트로서 조립 결합시켜주는 글루 컴포넌트를 제안하여, 인터페이스간 구문적 의미론적 불일치를 해결하였다.

그리고 Active Binding 컴포넌트들의 양측 인터페이스에 관한 메타데이터를 읽어 글루 컴포넌트를 자동 생성하는 조립도구 ArtComposer를 개발하였다. ArtComposer는 컴포넌트들을 시퀀스 다이어그램 형태로 조립하여 대규모 분산 시스템을 구축할 수 있게 한다. 컴포넌트 조립이 메소드 호출 단위로 시각화되므로 조립 작업은 물론 유지 보수 작업도 용이해진다.

7. 참고 문헌

[1] W.W. Agresti, “What are the new paradigms?,” New Paradigms for Software Development, pp. 6-10, IEEE, 1986.

[2] K. C. Kang, A Reuse-based software development methodology, Software reuse: emerging technology, IEEE Computer Society Press, Los Alamitos, CA, 1988

[3] D. Batory , S. O'Malley, “The design and implementation of hierarchical software systems with reusable components”, ACM Transactions on Software Engineering and Methodology (TOSEM), v.1 n.4, p.355-398, Oct. 1992

[4] N. Medvidovic, P. Oreizy, and R.N. Taylor, “Reuse of Off-the-Shelf Components in C2-Style Architectures”, Proc. 19th International Conference on Software Engineering, Boston, MA, pp. 692-700, 1997.

[5] Y. Choi, O. Kwon, G. Shin: An Approach to Composition of EJB Components using C2 Style, 28th Euromicro Conference (EUROMICRO'02), Dortmund, Germany, 2002.

[6] M. Fowler, “Inversion of Control Containers and the Dependency Injection pattern,” http://martinfowler.com/articles/injection.html.