컴포넌트 재사용을 지원하는 컴포넌트 모델 및 프레임워크

(A New Component Model and Framework for Reuse of Components)

임 윤 선 † 김 명 †† 정 승 남 ††† 정 안 모 ††††
(Yoonsun Lim) (Myung Kim) (Seungnam Jeong) (Anmo Jeong)

요 약 독립적으로 개발된 상용 컴포넌트들을 소스 코드 수정 없이 조립하여 재사용하기 위해서는 컴포넌트간 인터페이스 불일치 문제를 해결해야 한다. 본 논문에서는 이러한 문제를 해결하기 위한 새로운 컴포넌트 모델 및 조립 방법을 제안한다. 제안한 컴포넌트 모델은 각 컴포넌트가 서비스를 제공하는 하위 컴포넌트의 인터페이스에 의존하는 대신 서비스 요청에 관한 인터페이스를 독자적으로 정의하여 내장하고 이를 메타데이타로 노출한다. 조립 시에는 컴포넌트들의 요청 및 서비스 제공 인터페이스에 대한 메타데이타를 읽어 글루 컴포넌트 템플릿 코드를 자동 생성하여 인터페이스 정합에 필요한 중재 코드를 삽입하는 구조이다. 또 프레임워크에서 컴포넌트들에 대한 인스턴스를 관리하고 런타임시 의존성 주입 방식으로 조립을 수행하며, 글루 컴포넌트를 미들웨어 서비스 포인트 및 실시간 모니터링 포인트로 활용하는 방안도 제안한다. 이와 함께 본 논문은 제안한 컴포넌트 모델에 따르는 컴포넌트를 개발하고 조립하는 도구를 구현하여, 실용 가능성을 입증하였다.

키워드 : 컴포넌트 재사용, 컴포넌트 개발 방법론, 컴포넌트 모델, 컴포넌트 조립

Abstract It is difficult to assemble independently developed software components because of discrepancies between their interfaces. In order to resolve such problem, we propose a new component model, Active Binding Technology, in which each component has its own independently-defined interface for service request that is revealed in its metadata, instead of passively following the interface of a service-providing component. This model includes the use of the glue component, an interface mediating place, whose template code is automatically generated by reading in the metadata of the components to be combined. We also propose a runtime framework that holds the pool of component instances, completes the assembly of components in the manner of dependency injection, and performs middleware services and real-time system monitoring through glue components. In order to test the practical value of Active Binding Technology, we have made a tool, which supports the development and assembly of Active Binding components.

Key words : Component Reuse, CBD, Component Model, Component Assembly

† 학생회원 : 이화여자대학교 컴퓨터학과
lys96@ewhain.net
†† 종신회원 : 이화여자대학교 컴퓨터학과 교수
mkim@ewha.ac.kr
††† 정 회 원 : 리버넥스 개발실 이사
hellojsn@libernex.com
†††† 정 회 원 : 리버넥스 대표 대표이사
amjeong@libernex.com
논문접수 : 2007년 8월 16일
심사완료 : 2007년 11월 13일

정보과학회논문지: 소프트웨어 및 응용 제34권 제12호(2007.12)

1. 서 론

소프트웨어 전문가들은 대부분 이미 개발된 소프트웨어 모듈의 재사용이 소프트웨어 개발 생산성을 획기적으로 높여줄 것이라고 믿는다[1]. 이런 믿음의 대표적인 산물이 상용(COTS : Commercial Off-The-Shelf) 컴포넌트의 조립을 통한 소프트웨어 개발을 주창하며 1990년대에 출현한 컴포넌트 기반 개발 방법론(CBD: Component-based Development)이다[2]. 그러나 현실적인 여러 제약으로 인해 CBD 주창자들의 이상은 실현되지 못하였고, 재사용과 관련된 다양한 방법론들도 소프트웨어 모듈의 재사용과 손쉬운 조립을 실현하는 데

별다른 동력을 제공하지 못하였다[3,4].

대표적인 소프트웨어 모듈인 컴포넌트가 현실적으로 전혀 재사용되지 않는 것은 아니다. 예를 들어 닷넷 프레임워크와 같은 플랫폼이나 개발 툴에서 제공하는 라이브러리와 사용자 인터페이스 컨트롤들은 프로그래밍의 근간으로 재사용된다. 그러나 CBD 등에서 상정한 재사용 대상은 이와 같은 1차적 라이브러리가 아니라 그것들을 이용하여 실제적인 비즈니스 로직을 구현한 서비스 컴포넌트들이다. 플랫폼 라이브러리가 이미 주어진 개발 환경이라면, 새로운 소프트웨어를 개발하는 데 소요되는 노력의 대부분을 차지하는 것은 서비스 컴포넌트의 구현이며, 이를 이미 개발된 컴포넌트의 재사용으로 대체할 수 있다면 소프트웨어 개발 생산성이 대폭 향상되는 것은 자명하다. 그러나 기존의 컴포넌트 기술 기반으로는 서로 다른 개발자들에 의해 독립적으로 생산된 컴포넌트들을 소스 코드 수정 없이 조립하여 새로운 애플리케이션을 작성할 수 없으며, 이에 따른 개발 생산성 답보는 인터넷과 유비쿼터스 컴퓨팅 시대의 도래로 소프트웨어 수요가 폭증하는 상황에서 만성적인 소프트웨어 위기를 더욱 심화시키고 있다.

CBD 등에서 특별히 바이너리 컴포넌트의 재사용을 목표로 내세운 데는 이유가 있다. 이미 개발된 컴포넌트가 새로운 용도에 일부 부적합할 경우 소스 코드를 적절히 수정해주면 얼마든지 재사용할 수 있을 것이다. 그러나 독립적으로 개발된 컴포넌트의 소스 코드를 구할 가능성도 희박할 뿐더러, 타인이 작성한 소스 코드 분석과 수정에 소요되는 비용이 처음부터 새로 개발하는 것에 비해 크게 절감되는 것도 아니다. 따라서 시장에서 저렴한 비용으로 구할 수 있는 컴포넌트를 소스 코드를 수정하지 않고 재사용할 수 있어야 비로소 뚜렷한 생산성 향상 및 비용 절감 효과를 거둘 수 있는 것이다.

그러나 Java/EJB[5], .NET[6], COM+[7], CORBA/CCM[8]과 같은 기존 컴포넌트 기술들은 독립적으로 개발된 컴포넌트들을 소스 코드 수정 없이는 서로 조립하기 힘들다. 첫째 이유는 이 컴포넌트 모델들이 상호 인터페이스가 일치하지 않는 컴포넌트들을 연동시키는 방법을 제공하지 않기 때문이다. 두 번째 이유는 컴포넌트 식별자 문제이다. 기존 모델에서는 하위 컴포넌트 식별자를 이용하여 컴포넌트 객체를 생성하고, 메소드를 호출하는 코드가 상위 컴포넌트에 하드코딩(hard coding)되어 있어 인터페이스가 일치한다고 하더라도 새로운 버전의 하위 컴포넌트와 연동하려면 상위 컴포넌트의 코드를 수정해야 한다.

서비스 컴포넌트 재사용의 어려움으로 소프트웨어 개발 생산성을 향상시킬 수 있는 다른 방안들이 지속적으로 연구되고 있다. 그러나 도메인 엔지니어링을 수반하는 프로덕트 라인(product line)[9]이나 프레임워크 접근 방식(framework approach)은 일부 전문 분야에서만 재사용 효과를 발휘할 뿐, 모든 분야에 확대하여 적용되기는 부적합하다. 한편 최근 각광받고 있는 서비스 지향 아키텍처(SOA: Service-oriented Architecture)[10]도 재사용을 지원하는 모델이긴 하지만, 재사용 대상인 서비스는 비즈니스 관점의 업무 단위로 분할된 큰 규모의 모듈들이다. 따라서 각 서비스를 개발할 때는 개발 및 유지 보수의 효율성을 위해 어차피 객체 지향 방식으로 더 작게 분할되어 구현된 컴포넌트들을 재료로 써야 한다. 즉, 컴포넌트 재사용은 아직도 소프트웨어 위기를 타개할 유일한 해결책으로 그 실현이 절실히 필요하다.

본 논문은 기존의 컴포넌트 기술들이 왜 재사용을 효과적으로 지원하지 못하는가를 분석하고, 그 결함을 보정해줄 실용적인 방법을 고안하기 위하여 수행되었다. 본 논문에서는 상용 컴포넌트의 구입과 조립을 통한 소프트웨어 개발을 가능하게 하기 위하여 기존 컴포넌트 모델들과 다른 액티브 바인딩 기술(Active Binding Technology)이라는 명칭의 새로운 컴포넌트 모델을 제안한다. 이 컴포넌트 모델은 상하위 컴포넌트간 의존 관계를 글루 코드에서 정합하고 조립하는 기본 구조를 갖는다. 각 컴포넌트는 하위 컴포넌트에 정의된 인터페이스 규격을 따르는 대신 하위 컴포넌트 호출 규격을 독자적으로 정의하여 사용하고, 이에 대한 메타데이타를 외부로 노출시킨다. 즉 각 컴포넌트의 메타데이타에는 서비스 제공에 관한 인터페이스뿐 아니라 서비스 요청에 관한 인터페이스 정보가 포함된다. 컴포넌트 조립시에는 상위 컴포넌트의 서비스 요청 인터페이스와 하위 컴포넌트의 서비스 제공 인터페이스를 읽어들여 글루 코드를 자동으로 생성한다. 글루 코드에서 컴포넌트간 상호 일치하지 않는 인터페이스를 정합하도록 함으로써 독립적으로 개발되어 인터페이스가 다른 컴포넌트들을 소스 코드 수정 없이 유연하게 조립, 재사용할 수 있는 길을 열었다.

본 논문은 다음과 같이 구성된다. 2장에서 소프트웨어 컴포넌트 재사용 및 조립을 위한 기존 연구들을 살펴보고, 3장에서 본 논문에서 제안한 액티브 바인딩 컴포넌트 구조 및 조립 모델에 대해 설명한다. 4장에서는 액티브 바인딩 컴포넌트를 지원하는 액티브 바인딩 프레임워크에 대해 소개한다. 5장에서 본 논문에서 제안한 모델과 프레임워크를 구현한 조립 도구를 소개한 후, 6장에서 결론을 맺는다.

2. 기존 관련 연구

그동안 소프트웨어 컴포넌트 재사용 및 조립을 통한 애플리케이션 개발이라는 목표를 추구한 다양한 시도가 있었다. 그 대표적인 것으로 인터페이스의 구문론적, 의

미론적, 프로토콜 명세 확장을 통해 컴포넌트 어댑테이션(component adaptation)을 지원하는 형식적 방법(formal method)에 대한 연구들[11-14]과, 현존하는 다양한 컴포넌트 모델, 프로덕트 라인, 의존성 주입(dependency injection)[15], SOA 등을 들 수 있다.

2.1 컴포넌트 어댑테이션 관련 연구

지금까지 제3자가 개발하여 인터페이스가 상이한 바이너리 컴포넌트들을 서로 연동시키기 위하여 인터페이스 명세 확장을 통한 컴포넌트 어댑테이션을 목표로 하는 일련의 연구가 이루어져 왔다. 그 대표적인 예로 π-calculus 표현식을 사용하는 형식적 방법론[11,12]과, B formal method를 이용하는 방법[13]을 들 수 있다.

[11,12]에서 제안한 방법은 인터페이스 명세에 컴포넌트가 준수해야 할 상호작용의 프로토콜을 π-calculus 식으로 기술한 행위(behavior) 명세를 포함시키고, 컴포넌트 조립 시 인터페이스 간 메소드 및 매개변수 각각에 대한 중재와 더불어 메시지(메소드 호출) 순서를 적절히 중재하고 교착 상태를 회피할 수 있게 하는 어댑터를 만들며, 이를 검증하는 알고리즘을 제시하고 있다. 이에 비해 [13]은 인터페이스 데이타 모델, 불변식(invariant), 각 메소드의 선행 및 사후조건(pre/post condition) 등을 B-모델(B-Model)로 기술한 인터페이스 명세를 이용하여 컴포넌트 간 상호작용의 구문론적, 의미론적, 프로토콜 불일치를 해소하는 어댑터를 만드는 방법과 이 어댑터를 통한 컴포넌트 연동가능성(interoperability)의 검증 방법을 제시하고 있다.

그러나 이와 같은 종래의 컴포넌트 어댑테이션 연구들은 모두 컴포넌트와 어댑터의 명세 방법과 이 명세를 이용한 상호작용 검증 방법을 제시하고 있을 뿐, 명세에 대한 구체적인 구현 방법을 제시하지 않고 있다. 프로그래밍 언어와 비슷한 추상화 수준을 지니는 OCL이나 B-모델로 어떻게 간결하면서도 유용한 의미론적 행위 명세를 기술하고 검증할 것인가의 문제는 구문론적 불일치 중재 방법을 다루는 본 연구의 범위를 벗어나는 것이므로 논외로 친다 하더라도, 프로그램 언어와 매우 이질적인 π-calculus[16]나 B formal method[17] 등을 명세 수단으로 사용하고 있는 이런 방법론들이 산업적 실용성을 갖기 위해서는 명세 수단과 구현 언어 간의 이질성(번역상의 장애)을 효율적으로 극복할 수 있는 방법과 그것을 검증하는 방법이 제시되어야 하나 이에 대한 고려나 대안이 제시된 예는 없다. 또한 이러한 명세 수단의 채택은 컴포넌트 개발자나 어댑터를 개발해야 하는 조립자들에게 새롭고 어려운 명세 방법을 이해하고 정확히 사용해야 하는 부담을 지우면서도 저자들이 스스로 밝히고 있듯이 연동가능성에 대한 완전한 검증은 보장하지 못하므로 이들 연구는 단순한 예제들을 처리하는 수준의 실험실용 모델에 그칠 뿐, 산업에 곧바로 적용할 수 있는 실용적인 컴포넌트 재사용/조립 방법론으로 보기는 어렵다.

2.2 객체지향 컴포넌트 기반 기술

현대의 기업용 소프트웨어는 대부분 Java나 .NET 등 객체지향 컴포넌트 기술을 사용하여 다계층 구조의 분산 시스템으로 개발된다. 일반적으로 계층적 컴포넌트 모델에서는 최상위에 프리젠테이션 계층을, 중간에는 비즈니스 로직 계층을, 종단에는 데이타 서비스 계층을 배치한다. 이러한 다계층 구조에서 종단의 컴포넌트들을 제외한 나머지 대부분의 컴포넌트들은 인근 계층의 컴포넌트들과 인터페이스를 공유하면서 클라이언트-서버 관계로 결합되어 있으며, 각 컴포넌트는 자신의 서비스를 하위 계층 컴포넌트의 서비스에 의존하여 처리한 후 상위 계층의 컴포넌트에 제공한다.

Java/EJB, .NET, COM+, CORBA/CCM과 같은 기존 컴포넌트 모델을 따르는 컴포넌트들은 하위 컴포넌트에 대한 서비스 호출이 정적으로 코딩된 채로 컴파일되어 블랙박스 형태로 배포된다. 이들은 자신이 제공하는 서비스에 대한 인터페이스는 메타데이타로 외부에 노출하지만 자신이 요구하는 서비스에 대한 인터페이스는 따로 정의하여 노출시키지 않으며, 하위 컴포넌트와의 연동은 철저히 하위 컴포넌트의 인터페이스에 따른 코딩을 통해 구현된다[18]. 이렇게 콜스택(call stack) 방식으로 하위 계층 컴포넌트와 밀접하게 결합된 컴포넌트들은 원래 사용되었던 애플리케이션 외의 환경에서는 소스 코드 수정 없이 재사용되기 어렵다. 서로 다른 독립 벤더들이 시장의 수요를 예상하여 개발한 컴포넌트들이 기능적 조화는 이루고 있다고 해도 동일한 인터페이스를 사용할 가능성은 거의 없기 때문이다. 또한 컴포넌트 개발자들이 소스 코드를 함께 제공하지도 않을 것이고, 설사 소스 코드를 구할 수 있다고 해도 타인이 작성한 코드를 분석하여 수정하는 것은 컴포넌트를 독립적으로 개발하는 것보다 더 효율적인 것도 아니다.

소스 코드 수정이 불가능한 컴포넌트의 경우 클라이언트와 서버 사이의 상이한 인터페이스를 정합하는 중재 코드를 삽입함으로써 서비스를 주고받게 할 수 있다. Java, .NET 등은 컴포넌트로부터 메타데이타를 읽어낼 수 있는 리플렉션(reflection) 기능을 제공한다[19,20]. 이 기능을 이용하면 특정 컴포넌트가 상위 컴포넌트에 제공하는 서비스의 인터페이스 정보를 상세히 읽어낼 수 있다. 그러나 자신이 요청하는 서비스에 대해서는 그 서비스를 제공하는 하위 컴포넌트의 이름만 알아낼 수 있을 뿐이다. 즉, 하위 컴포넌트가 제공하는 인터페이스 중에서 어떠한 인터페이스와 메소드를 호출하는가, 그 메소드의 시그너처는 무엇인가, 하위 컴포넌트의 메소드

를 언제 호출하는가 등 인터페이스 정합에 필요한 상세 정보는 전혀 알 수 없다. 부족한 정보는 컴포넌트 호출에 사용되는 인터페이스와 의미적 행위(semantic behavior)를 상세히 기록한 문서를 통해 얻을 수밖에 없다. 그러나 기술법상의 표준이 전혀 없는 상황에서 컴포넌트 개발자가 이러한 정보를 오해의 소지 없이 명료하게 작성하기는 매우 어렵다.

컴포넌트에 대해 상세히 기록한 문서를 구할 수 있다고 해도 중재 코드를 수작업으로 일일이 코딩하고 관리하는 것은 쉬운 일이 아니다. 더욱이 기존 모델에서는 하위 컴포넌트 식별자를 이용하여 컴포넌트 객체를 생성하고 메소드를 호출하는 코드가 상위 컴포넌트에 하드 코딩되어 있다. 따라서 인터페이스가 일치한다고 해도 새로운 버전의 하위 컴포넌트와 연동하려면 상위 컴포넌트의 코드를 수정해야 하지만, 시장에서 구한 상하위 컴포넌트 사이에 컴포넌트 식별자가 일치할 가능성은 매우 희박하다. 또한 기능적으로 용도에 맞는 상용 컴포넌트들이 시장에 풍부하게 존재한다 하더라도 기존 컴포넌트 기술 기반으로는 이들을 소스 코드 수정 없이 조립하여 소프트웨어를 개발할 방법이 없는 것이나 다름없다.

2.3 객체간 의존성 제거를 위한 디자인 모델

클라이언트 컴포넌트와 서비스 구현체 사이의 의존성을 제거하는 디자인 모델로는 의존성 주입, 서비스 로케이터 (service locator) 등이 있다. 이 중 의존성 주입은 Spring[21], PicoContainer[22]와 같은 다수의 경량 컨테이너들이 서로 다른 객체들을 연결하기 위한 방법으로 사용하고 있다.

의존성 주입의 기본 개념은 객체들을 연결해 주는 별도의 객체를 갖는 것이다. 즉, 클라이언트 컴포넌트가 서버 컴포넌트 객체를 직접 인스턴스화 하는 대신, 컴포넌트 컨테이너가 서버 컴포넌트를 인스턴스화하고, 그 레퍼런스를 클라이언트 컴포넌트의 속성이나 생성자 매개변수로 주입하는 것이다. 이런 방식을 따르면 상위 컴포넌트의 클래스가 하위 컴포넌트의 구현 클래스를 컴파일 시점에 알 필요가 없고, 하위 컴포넌트 식별자를 가질 필요가 없게 되므로 컴포넌트 조립이 유연해진다. 그러나 이 모델 역시 호출/피호출 인터페이스가 일치하는 컴포넌트들만 조립할 수 있을 뿐, 독립적으로 개발되어 인터페이스가 상이한 컴포넌트들의 조립은 지원하지 않으므로 실질적인 컴포넌트 재사용 효과는 거의 없다고 할 수밖에 없다.

3. 액티브 바인딩 컴포넌트 모델

인근 계층의 컴포넌트들과 클라이언트-서버 관계를 형성하면서 동작하도록 개발된 컴포넌트들을 소스 코드 수정 없이 독립적으로 재사용하는 것을 현실화하기 위해서는 우선 컴포넌트들이 특정 애플리케이션의 아키텍처에 종속되지 않도록 하는 체계가 필요하다. 즉 플랫폼이나 컴포넌트 프레임워크에 대한 종속성은 피할 수 없다고 하더라도 컴포넌트간의 종속성은 완전히 해소되어야 한다. 둘째, 컴포넌트 조립 비용이 컴포넌트 개발 비용에 비해 현저하게 저렴해야 한다. 이를 위해서는 비주얼 모델 방식의 조립 방법과 글루 컴포넌트 코드(glue component code)를 생성해주는 자동화 도구와 같이 컴포넌트들을 쉽고 신속하게 조립할 수 있는 환경이 필요하다. 셋째, 기업용 애플리케이션 아키텍처에서 일반적으로 사용되는 다계층 분산시스템 설계에서 얻은 경험과 최신 상용 컴포넌트 기반 기술들이 이음새 없이 조화를 이루어야 한다. 본 논문은 이상의 조건들을 충족시키기 위한 컴포넌트 구조와 개발 프로시저, 조립 방법과 컴포넌트 프레임워크, 자동화 도구를 포함하는 새로운 컴포넌트 모델을 제시한다.

3.1 액티브 바인딩 컴포넌트 구조

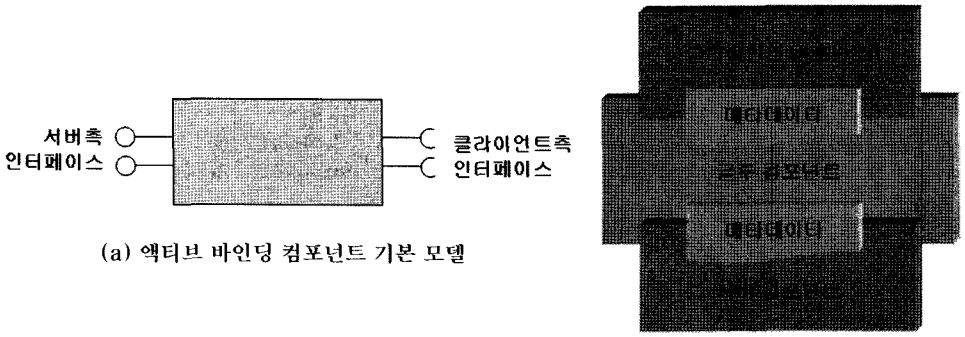
액티브 바인딩 컴포넌트는 필요한 하위 컴포넌트의 서비스 명세를 외부로 노출하는 기본 구조를 갖는다. 즉 그림 1(a)과 같이 컴포넌트가 제공하는 서비스 명세인 서버측 인터페이스와 하위 컴포넌트의 서비스 명세인 클라이언트측 인터페이스를 갖는 구조이다. 또한 액티브 바인딩 컴포넌트는 그림 1(b)와 같이 하위 컴포넌트와 밀결합되는 것을 피하기 위해 하위 컴포넌트 객체의 메소드를 직접 호출하는 대신 조립도구가 자동으로 생성하는 글루 컴포넌트를 통해 하위 컴포넌트와 결합되는 구조는 갖는다. 본 논문에서는 서비스를 제공하는 컴포넌트를 서버 컴포넌트, 서비스를 받는 컴포넌트를 클라이언트 컴포넌트라고 부른다.

그림 2는 .NET 컴포넌트(닷넷 용어로는 어셈블리)를 확장하여 만든 액티브 바인딩 컴포넌트의 구조이다. .NET 컴포넌트는 컴포넌트 차원의 메타데이타인 메니페스트(manifest), 컴포넌트가 구현한 인터페이스와 클래스 정보로 이루어진 타입 메타데이타, 컴파일된 실행 코드, 비트맵이나 스트링 같은 리소스로 구성된다. 액티브 바인딩 컴포넌트는 메니페스트에 글루 컴포넌트의 식별자를 추가 포함하고, 타입 메타데이타에는 클라이언트 측 인터페이스 정보와 구현 클래스 이름을 추가적으로 포함한다.

액티브 바인딩 컴포넌트를 구현하는 알고리즘은 다음과 같다.

[단계1] 컴포넌트를 개발하는 과정에서 하위 컴포넌트의 서비스가 필요한 경우 하위 컴포넌트에 정의된 인터페이스에 의존하는 대신 클라이언트측 인터페이스를 독자적으로 정의한다.

[단계2] 클라이언트측 인터페이스를 구현할 컴포넌트 및



(a) 액티브 바인딩 컴포넌트 기본 모델

(b) 액티브 바인딩 컴포넌트 결합 모델

그림 1 액티브 바인딩 컴포넌트 모델

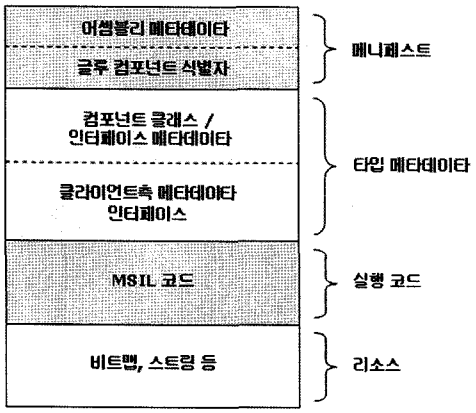


그림 2 .NET 컴포넌트를 확장한 액티브 바인딩 컴포넌트 구조

클래스 식별자(identifier)를 생성하고 사용자 정의 애노테이션(custom annotation)이나 애트리뷰트(custom attribute)를 이용해 단계 1에서 정의한 인터페이스에 부가한다.

[단계3] 컴파일과 테스트를 위해 단계 1과 단계 2에서 정의된 클라이언트측 규격에 따르는 스터브(stub) 컴포넌트를 생성하여 참조 라이브러리(reference library)에 추가한다.

[단계4] 개발 중인 컴포넌트에 상기 스터브 컴포넌트 객체를 생성하고 그 메소드를 호출하는 코드를 삽입한다.

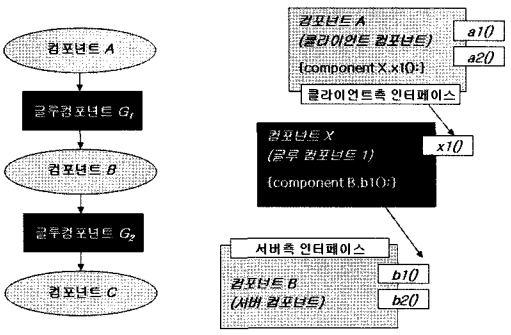
[단계5] 스터브 컴포넌트 코드를 편집하여 개발 중인 컴포넌트를 테스트한다.

스터브 컴포넌트는 개발과 테스트 목적으로 사용될 뿐 추후 조립 시에는 글루 컴포넌트로 대치된다.

3.2 액티브 바인딩 컴포넌트 조립 구조

액티브 바인딩 컴포넌트를 조립하기 위한 방법은 다음과 같다. 그림 3(a)와 같이 서로 다른 개발자들에 의해 독립적으로 개발되어 상이한 인터페이스를 갖는 액티브 바인딩 컴포넌트 A, B, C가 있고, 이들을 조립해야 할 때, 글루 컴포넌트 G_1과 G_2를 생성하여 호출/피호출 인터페이스 불일치를 중재함으로써 조립을 완성한다.

독립적으로 개발되어 인터페이스가 일치하지 않는 액티브 바인딩 컴포넌트 A와 B를 조립하는 방법은 그림 3(b)에 있다. 그림에서와 같이 클라이언트 컴포넌트 A가 서버 컴포넌트 B의 인터페이스가 제공하는 서비스 $b_1()$을 호출해야 하는 경우, 클라이언트 컴포넌트 A는 서버 컴포넌트 객체 B의 식별자나 b_1의 시그너처에 관계없이 독자적으로 임의의 클라이언트 측 인터페이스를 정의하여 사용하고 있고, 이에 대한 상세한 정보를 메타데이타로 포함하고 있으므로, 조립도구가 리플렉션 기능을 이용해 이 메타데이타를 읽어 클라이언트 컴포넌트에 정의된 인터페이스를 구현한 글루 컴포넌트 객체를 만든다. 조립도구는 이렇게 생성된 글루 컴포넌트의 메소드 $x_1()$에서 서버 컴포넌트 B의 인터페이스 $b_1()$을 호출하여 결과 값을 클라이언트 컴포넌트 A로 전달함으로



(a) 액티브바인딩 컴포넌트 조립모델 (b) 액티브바인딩 컴포넌트 조립 방법

그림 3 액티브 바인딩 컴포넌트 조립

써 컴포넌트 A가 서버 컴포넌트 B의 서비스를 클라이언트 호출하여 사용할 수 있게 한다.

액티브 바인딩 기술에서는 애플리케이션의 각 유스케이스에 관여하는 컴포넌트들의 상호작용에 대한 전체적인 뷰를 UML 순차도 형태의 인터랙션 모델로 표현하고 이를 조립에 이용한다. 액티브 바인딩 컴포넌트 조립 방법을 조립자의 역할과 조립도구의 역할로 구분하여 좀 더 상세히 설명하면, 우선 조립자는 조립해야 하는 유스 케이스에 관여하는 컴포넌트들을 GUI 조립 도구를 이용하여 배치하고 컴포넌트간 호출/피호출을 표현하는 화살표 형태의 메시지를 그어 인터랙션 모델을 완성한다. 여기서 각 메시지는 메소드 호출/피호출을 중재하는 글루 컴포넌트와 매핑된다. 이어서 조립자가 메시지의 속성창을 열면 조립도구가 클라이언트 컴포넌트로부터 클라이언트측 인터페이스 정보를, 서버 컴포넌트로부터 서비스 인터페이스 정보를 읽어 보여준다. 조립자가 서로 연결될 호출 메소드와 피호출 메소드를 선택하면 조립도구는 읽어들인 메타데이타를 이용해 클라이언트측 인터페이스를 구현한 글루 컴포넌트와 그 객체, 메소드 코드를 생성하고, 생성된 메소드 내에 서버 컴포넌트의 피호출 메소드를 호출하는 코드를 생성하여 삽입한다.

조립자는 이렇게 생성된 메소드 코드를 편집하여 두 컴포넌트간의 구문적 불일치를 중재하는 것 외에도 컴포넌트 사이의 매개 변수들 사이에 존재할 수 있는 의미론적 부정합을 해결할 수 있다. 예를 들어 클라이언트 컴포넌트는 화폐 단위 원을 서버 컴포넌트에 매개 변수로 전달하는데 서버 컴포넌트는 달러를 매개 변수로 받도록 되어 있는 경우 조립자가 글루 컴포넌트에 원을 달러로 변환하는 코드를 삽입하여 이를 중재할 수 있다.

따라서 액티브 바인딩 컴포넌트를 개발할 때 클라이언트측 인터페이스와 서버측 인터페이스 메소드에 커스텀 애트리뷰트를 이용하여 상세한 설명을 부가해 놓으면, 컴포넌트들을 조립할 때 조립자가 적절한 호출/피호출 메소드를 선택하는 데 큰 도움을 받을 수 있다. 또한 컴포넌트의 각 서비스 메소드에 그 메소드가 의존하는 클라이언트측 인터페이스 식별자들을 의존 순서에 따라 메타데이타로 부가하는 방식으로 조립 시에 유용하게 참조될 행위계약(behavioral contract) 정보를 기술할 수 있다.

4. 액티브 바인딩 컴포넌트 프레임워크

액티브 바인딩 기술에서는 컴포넌트들 사이의 메소드 호출/피호출을 반드시 글루 컴포넌트가 중재하도록 한다. 이 글루 컴포넌트는 상이한 인터페이스를 중재하여 독립적으로 개발된 컴포넌트들이 상호 조립될 수 있도록 하는 데에 그 일차적인 목적이 있으나, 모든 컴포넌트 객체의 서비스 메소드 호출이 반드시 글루 컴포넌트를 거쳐 이루어지므로 비즈니스 도메인 컴포넌트 객체들에 대하여 적용해야 하는 횡단기능(cross-cutting concerns)을 부가하기에 최적의 장소이기도 하다. 이처럼 조립 시에 자동으로 생성되는 글루 컴포넌트를 횡단기능을 구현한 애스펙트(aspect) 부가 장소로 사용하는 액티브 바인딩 조립 메커니즘은 기존 AOP(Aspect-Oriented Programming)의 위빙(weaving) 및 포인트컷(pointcut) 메커니즘과 달리 컴포넌트 객체의 소스 코드나 새로운 구문의 AOP 언어를 필요로 하지 않는 매우 효율적인 AOP 프레임워크의 구축을 가능하게 한다.

액티브 바인딩 기술에서는 컴포넌트에서 독자적으로 클라이언트측 인터페이스를 정의하여 사용하고, 이를 구현한 글루 컴포넌트 코드가 조립 시에 동적으로 생성되는 방식은 유지하되, 글루 컴포넌트의 객체 생성과 참조 설정을 프레임워크에 위임하는 의존성 주입 방식을 쓰며 이는 그림 4에 나타나 있다. 이 방식을 사용하면 런타임에 프레임워크의 개입이 수반되어야 하지만 여러 버전의 글루 컴포넌트 사용이 가능해진다. 또한 객체들의 생명주기와 객체 풀 관리를 프레임워크가 담당하게 함으로써 실행 속도가 빨라지고 컴퓨터 자원을 효율적으로 사용할 수 있다.

그림 5는 본 논문에서 엔터프라이즈급 QoS(Quality of Service)를 지원하는 SOA 서비스 미들웨어 역할을 수행하도록 설계된 액티브 바인딩 컴포넌트 프레임워크의 구조이다. 이 모델은 프레임워크가 컴포넌트들의 인스턴스 관리자 및 의존성 주입 역할을 수행함과 동시에 애스펙트로 구현된 미들웨어 서비스를 플러그인할 수 있는 구조로서, 비즈니스 도메인 컴포넌트 객체들의 조립은 물론 이들 객체들과 QoS 관련 각종 애스펙트의 자유로운 조합을 지원한다. 또한 글루 컴포넌트에서 컴포넌트들 사이에 전달되는 매개변수와 리턴 값, 응답 시간 등을 읽어 이들의 값이 허용 범위 안에 드는지 여부를 체크함으로써 시스템의 건강 상태를 실시간 모니터링할 수도 있다.

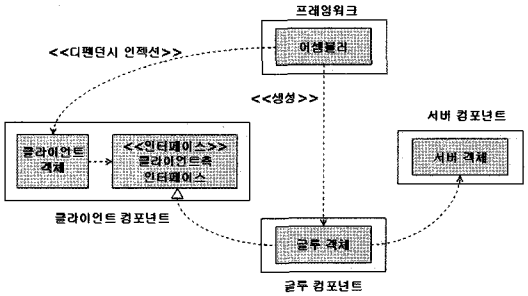


그림 4 액티브 바인딩 프레임워크의 의존성 주입

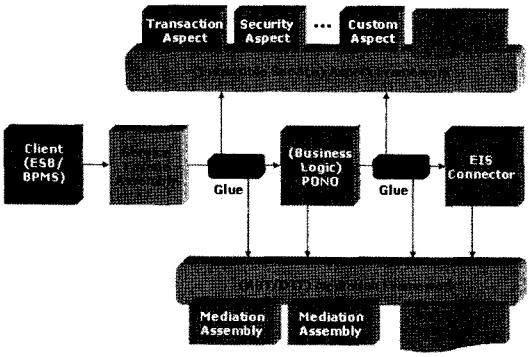


그림 5 액티브 바인딩 프레임워크 아키텍처

5. 액티브 바인딩 컴포넌트 개발 및 조립도구 구현

5.1 컴포넌트 조립도구 - 아트컴포저

본 논문에서 제안한 액티브 바인딩 기술의 실용성을 검증하기 위해 액티브 바인딩 컴포넌트의 개발과 조립을 지원하는 아트컴포저(ArtComposer)를 개발하였다. 아트컴포저는 글루 컴포넌트 생성 마법사를 제공한다. 개발자는 하위 컴포넌트가 제공하는 서비스가 필요할 때 마법사를 이용하여 호출 시그너처를 입력한다. 아트컴포저는 개발자가 입력한 정보를 바탕으로 클라이언트 측 인터페이스, 글루 컴포넌트와 구현 클래스의 식별자, 행동계약 정보, 글루 컴포넌트 객체 생성 및 호출에 필요한 코드 등을 자동 생성함으로써, 액티브 바인딩 컴포넌트를 매우 효율적으로 개발할 수 있게 한다. 아트컴포저는 또한 그림 6에서와 같이 컴포넌트들을 순차도 형태로 조립할 수 있는 그래픽 사용자 인터페이스를 지원한다. 소프트웨어 개발자가 이 도구에 조립할 컴포넌트들을 배치하고 컴포넌트들 사이의 메시지 흐름을 설정하면 각 컴포넌트 쌍을 연결하는 글루 컴포넌트가 자동 생성되며, 인터페이스 정합이 필요한 경우 개발자는 글루 컴포넌트에 중재 코드를 삽입할 수 있다.

5.2 컴포넌트 조립의 실용성 검증

본 논문은 다음과 같은 실험을 통하여 액티브 바인딩 컴포넌트 모델이 독립적으로 개발된 컴포넌트들의 조립을 실제로 가능케 하는 것을 확인하였다. 먼저 간단한 수강신청 프로그램의 아키텍처를 그림 7과 같이 3계층으로 설계한 다음 각 계층의 컴포넌트들을 식별하고 각 컴포넌트가 지원해야 할 기능 리스트를 만들었다. 이어서 실험에 참여한 학생들을 여러 팀으로 나누어 각각의 컴포넌트를 독립적으로 개발하게 하였다. 이 때 인터페이스 세부 규격은 각 팀이 독자적으로 설계하되 다른

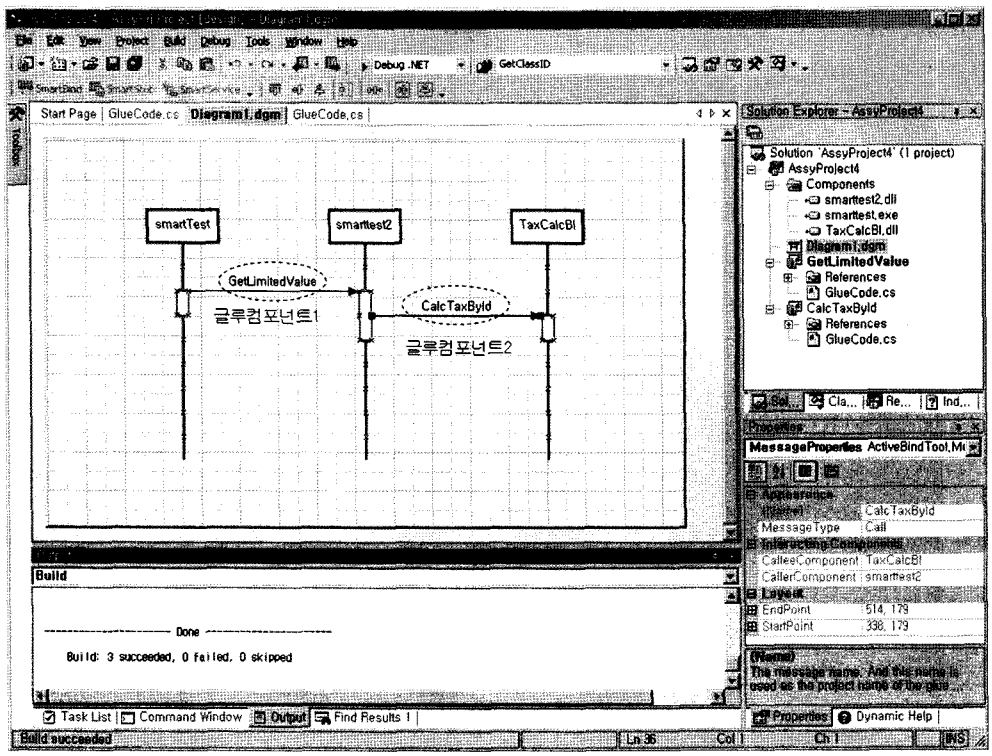


그림 6 아트컴포저를 이용한 컴포넌트 조립화면

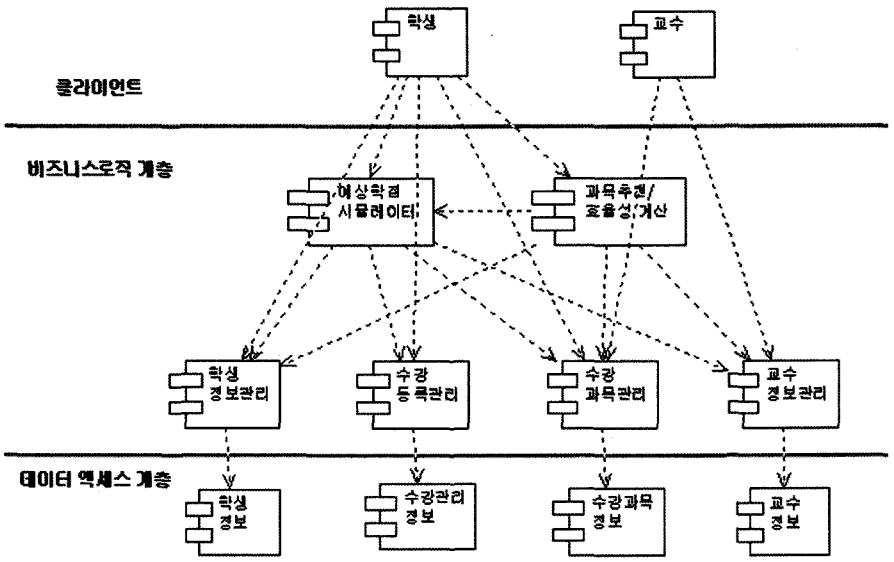


그림 7 아트컴포저를 이용하여 조립한 수강신청 프로그램 컴포넌트 다이어그램

팀과는 상의할 수 없게 하여 컴포넌트간의 인터페이스 불일치를 의도적으로 조장하였다. 본 논문은 이렇게 만들어진 액티브 바인딩 컴포넌트들을 조립하여 수강신청 프로그램을 만들고 그 동작을 확인하였다.

5.3 컴포넌트 조립의 효율성 검증

액티브 바인딩 조립에서 생성되는 글루 컴포넌트가 성능에 미치는 영향을 측정하기 위해 다음과 같은 실험을 수행하였다. 클라이언트 컴포넌트는 레코드 개수와 지연시간을 서버 컴포넌트에 매개변수로 전달한다. 서버 컴포넌트는 5개의 스트링 타입 필드와 3개의 정수 타입 필드, 2개의 데이타 타입 필드로 이루어지고 있는 526 바이트 데이타 전송 객체(DTO : Data Transfer Object)를 클라이언트 컴포넌트로부터 매개변수로 받은 레코드 개수만큼 만들어 클라이언트 컴포넌트에 결과값으로 전달한다. 클라이언트 컴포넌트에서 전달한 지연시간은 서버 컴포넌트의 비즈니스 로직 처리시간을 가정한 것이다. 서버 컴포넌트는 클라이언트 컴포넌트에서 설정한 지연시간이 지난 후 레코드 셋을 클라이언트 컴포넌트에게 전달한다. 실험은 펜티엄(Pentium) 4, 3GHz CPU를 장착한 PC에서 닷넷 프레임워크 2.0을 이용하여 개발하였다. 본 실험은 다음 3가지 조립 방법에 대해 응답시간을 비교 측정하였다.

- 직접호출(DC : Direct Call) 조립방법 : 기존의 컴포넌트 조립 방법으로 직접 클라이언트 컴포넌트에서 서버 컴포넌트의 메소드를 호출한다.
- 액티브바인딩기술-참조에 의한 호출(ABT-CBR : Active Binding Technology-Ca By Reference) 조립방법 : 액티브 바인딩 컴포넌트 모델에 따라 만든 클라이언트 컴포넌트는 글루 컴포넌트를 통해 서버 컴포넌트의 메소드를 호출한다. 이때 글루 컴포넌트는 단지 서버 컴포넌트를 호출하고, 서버 컴포넌트로부터 전달 받은 레코드 셋의 레퍼런스를 클라이언트 컴포넌트에 전달하는 역할을 한다. 즉, 글루 컴포넌트에 의한 1개의 메소드 호출이 성능에 어느 정도의 부담을 초래하는 가를 알아보기 위한 조립 모델이다.
- 액티브바인딩기술-값에 의한 호출(ABT-CBV : Active Binding Technology-Call By Value) 조립방법 : ABT-CBR 조립방법과 마찬가지로 액티브 바인딩 컴포넌트 모델에 따라 조립을 하나, 이 경우는 글루 컴포넌트에서 서버 컴포넌트에서 전달받은 결과값을 수정하여 클라이언트 컴포넌트에 전달하는 조립 방법이다. 즉 글루 컴포넌트에서 인터페이스 불일치를 중재하기 위해 매개변수 및 결과값을 복사할 때, 성능에 미치는 영향을 분석하기 위한 조립 모델이다.

표 1 레코드개수 30개(15.4KB)일 경우 조립 방법에 따른 응답시간 측정

조립방법 \ 지연시간(ms)	1	10	100
DC	2.031	10.781	100.625
ABT-CBR	1.875	10.781	100.625
ABT-CBV	2.031	10.625	100.625

표 2 레코드개수 3000개(1.5MB)일 경우 조립 방법에 따른 응답시간 측정

조립방법 \ 지연시간(ms)	1	10	100
DC	5.000	13.750	103.750
ABT-CBR	5.000	13.750	103.594
ABT-CBV	5.156	13.906	103.750

일반적인 엔터프라이즈 애플리케이션 소프트웨어에서는 서버 컴포넌트에 데이타 검색 요청 시 보통 한번에 30개 내외의 레코드 수를 요구한다. 표 1은 글루 컴포넌트를 거쳐 가는 레코드 개수를 30개, 즉 데이타 크기를 약 15.4KB로 설정하고, 비즈니스 로직의 처리 지연 시간을 1ms~100ms까지 변화시켰을 때, 각 조립 방법에 따라 응답시간을 측정한 결과이다. 측정 결과를 살펴보면, 지연 시간이 1ms인 경우 ABT-CBR 조립방법의 응답시간이 1.875ms로 다른 조립방법에 비해 0.156ms 빠르게 나타났고, 지연시간이 10ms인 경우는 ABT-CBV 조립방법이 0.156ms 빠른 응답시간을 보였다. 그러나 지연시간이 100ms인 경우는 모든 조립방법이 동일한 응답시간 결과가 나왔다.

표 2는 의도적으로 글루 컴포넌트에 의한 부하를 늘리기 위해 클라이언트 컴포넌트에 전달할 결과값인 레코드 개수를 3000개, 즉 데이타의 크기를 약 1.5MB로 만들어 실험한 결과 값이다. 표 1과 마찬가지로 비즈니스 로직의 처리 지연 시간을 1ms~100ms까지 변화시키며, 각 조립 방법에 따라 응답시간을 측정하였다. 그 결과, 지연 시간이 1ms인 경우와 10ms인 경우 ABT-CBV 조립방법의 응답시간이 다른 조립방법에 비해 0.156ms 느리게 나타났고, 지연시간이 100ms인 경우는 DC 조립방법과 ABT-CBV 조립방법이 0.156ms 느린 응답시간을 보였다.

위의 실험 결과, 주어진 모든 조건에서 DC 조립방법과 ABT-CBR 조립방법의 응답 시간이 같거나, 1 타임 틱(time tick) 범위 내에서 DC 조립방법이 빠르거나 ABT-CBR 조립방법이 빠르게 나타났다. 이 실험 결과는 닷넷 프레임워크 등의 현대 런타임 환경이 메소드 호출을 매우 효율적으로 처리하므로 글루 컴포넌트를 통한 조립에 의해 한번의 메소드 호출이 추가된다 할지라도 그 성능 부담은 무시할 수 있는 수준이라는 것을 보여준다. ABT-CBV 조립방법의 실험결과는 글루 컴포넌트에서 매개변수나 결과값의 복사 부담이 발생할 경우 성능에 미치는 영향을 보인다. 복사 부담의 정도를 높이기 위해 매 호출마다 레코드 3000개, 즉 약 1.5MB의 데이타 복사가 일어나게 만들어서 실험한 경우에도 ABT-CBR 조립방법에 의한 성능 부담은 1.1%에 불과했다.

6. 결 론

본 논문에서는 서로 다른 개발자들에 의해 독립적으로 개발되어 인터페이스가 상이한 컴포넌트들을 소스 코드 수정 없이 조립할 수 있도록 하는 새로운 컴포넌트 모델을 제안하고 이 모델을 적용한 컴포넌트 개발 및 조립 도구를 구현하였다. 또한 제안한 방법에 따라 독립적인 인터페이스를 갖는 컴포넌트들을 개발하고, 조립하는 실험을 통해 액티브 바인딩 컴포넌트 모델과 자동화 도구가 그 목적을 효율적으로 달성하는 것을 확인하였다. 또 성능 실험을 통해 글루 컴포넌트로 인한 성능 부담이 무시할 정도라는 것도 확인하였다. 한편 본 논문에서는 개발자들에게 학습 부담과 오류를 유발할 수 있는 완전히 새로운 컴포넌트 모델을 만드는 대신 기존의 최신 상용 컴포넌트 모델과 자동화 도구를 확장함으로써 완성도 높은 개발 효율성을 유지하면서 재사용성과 유지 보수 효율성을 높이는 방안을 제시하였다.

컴포넌트가 소스 코드 수정 없이 재사용될 수 있으려면 서비스 제공 인터페이스와 서비스 요청 인터페이스를 모두 갖추어야 한다는 것은 새로운 발상이 아니다. 또 상이한 인터페이스를 정합하기 위해 글루 컴포넌트가 사용될 수 있다는 것도 새로운 아이디어가 아니다. 본 연구의 신규성과 독창성은 종래의 연구들이 제시한 재사용 관련 아이디어들이 구체적인 구현 방안 없이 순수한 개념 내지 명세 차원에 머물러 있는 것과 달리 서비스 요청 인터페이스를 누가 어떻게 정의하고, 어디에 위치시키며, 언제 어떻게 사용하는가, 또한 인터페이스를 정합하는 글루 컴포넌트는 어떻게 생성되며 수정되는가를 명확히 규정한 데에 있다. 다시 말해서 본 연구는 실제 소프트웨어 생산 환경에 적용 가능한 명확하고 실질적인 명세 및 구현 방법을 제시하고, 이를 지원하는 도구를 통해 그 실용성을 검증하였다는 점에서 여타 연구들과 차별화된다고 할 수 있다.

본 논문에서 제안한 액티브 바인딩 기술은 독립적으로 개발된 컴포넌트의 소스 코드 수정 없는 재사용을 실천할 수 있는 기반을 제공한다. 그러나 상용 컴포넌트를 구입하여 "조립에 의한 개발(development by assembly)" 방식으로 애플리케이션을 개발하는 CBD의 이상을 제대로 달성하기 위해서는 컴포넌트의 부품화 및 그 조립 방안과는 별도로 개발자가 필요로 하는 컴포넌트들의 신속한 검색과 자동 조립을 지원할 수 있도록 컴포넌트의 기능과 규격 정보를 정형화하여 표현하는 방법이 더 연구되어야 한다. 또 컴포넌트의 재사용율을 더욱 높이기 위해서는 애플리케이션마다 달라지는 비즈니스 데이타 객체에 대한 서비스 컴포넌트의 종속성을 제거하는 연구도 필요하다.

참 고 문 헌

[1] B. W. Boehm, M. H. Penedo, E. D. Stuckle, R. D. Williams and A.B.Pyster, "A Software Development Environment for Improving Productivity," IEEE Computer, Vol.17, No.6, pp. 30-44, June, 1984.

[2] P. Clements, "From Subroutines to Subsystems: Component-Based Software Development," The American Programmer, Vol.8, No.11, November, 1995.

[3] Steve Latchem, "Component Infrastructures: Placing Software Components in Context," Chapter 15 of Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley, 2001.

[4] Antonio Vallecillo, Juan Hernandez and Jose M. Troya, "New Issues in Object Interoperability," In Object-Oriented Technology: ECOOP 2000 Workshop Reader, No. 1964 in LNCS, pp. 256-269, Springer Verlag, 2000.

[5] Richard Monson-Haefel. "Enterprise JavaBeans," O'Reilly, Second edition, 2000.

[6] "An Introduction to Microsoft .NET. White Paper," Microsoft Corporation, 2001.

[7] Dale Rogerson, "Inside COM: Microsoft's Component Object Model," Microsoft Press, 1997.

[8] Object Management Group, "CORBA Component Model Specification," April 2006.

[9] J. Bosch, "Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach," Addison-Wesley, Boston, 2000.

[10] Thomas Erl, "Service-Oriented Architecture(SOA): Concept, Technology, and Design," Prentice Hall, 2005.

[11] Andrea Bracciali, Antonio Brogi and Franco Turini, "Coordinating Interaction Patterns," In Proceedings of ACM Symposium on Applied Computing, Las Vegas, United States, 2001.

[12] Andrea Bracciali, Antonio Brogi and Carlos Canal, "Systematic component adaption," In Proceedings of Workshop on Formal Methods and Component Interaction (FMCI 2002), Málaga, Spain, pp. 340-351, 2002.

[13] Inès Mouakher, Arnaud Lanoix and Jeanine Souquières, "Component Adaptation: Specification and Verification," In 11th International Workshop on Component Oriented Programming, Nantes, France, 2006.

[14] R. H. Reussner, "Adapting Components and Predicting Architectural Properties with Parameterised Contracts," In Tagungsband des Arbeitstreffens der GI Fachgruppen 2.1.4 und 2.1.9, Bad Honnef, W. Goerigk, Ed., 2001, pp. 33-43.

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

[16] Robin Milner, "The polyadic π-calculus: a tutorial," Technical report, University of Edinburgh, 1991.

[17] Jean R. Abrial, "The B Book : Assigning Programs to Meanings," Cambridge University Press, 1996.

[18] K. Whitehead, "Component-based Development: Principles and Planning for Business Systems," Addison-Wesley, 2002.

[19] Ira R. Forman and Nate Forman, "Java Reflection in Action," Manning, 2005.

[20] Hari Shankar, "Reflection in .NET," http://www.c-sharpcorner.com, February 26, 2001.

[21] Bruce A. Tate and Justin Gehtland, "Spring: A Developer's Notebook," O'REILLY, 2005.

[22] PicoContainer, "Core Concepts : Dependency Injection," http://www.picocontainer.org/injection.html.

임 윤 선

1987년 중앙대학교 수학과 학사. 2003년 이화여자대학교 컴퓨터학과 석사. 2004년~현재 이화여자대학교 컴퓨터학과 박사 과정. 관심분야는 온톨로지, 소프트웨어 재사용, 지식공학 등

김 명

1981년 이화여자대학교 수학과 학사. 1983년 서울대학교 계산통계학과 석사. 1993년 캘리포니아 주립대학교 (산타바바라) 컴퓨터학과 박사. 1993년~1994년 캘리포니아 주립대학교(산타바바라) 컴퓨터학과 Postdoc, 강사. 1995년~현재 이화여자대학교 컴퓨터학과 교수. 관심분야는 소프트웨어 재사용, 지식공학, 스트림 데이타처리, 온톨로지, 고성능 컴퓨팅 등

정 승 남

1984년 한양대 영어영문(학사). 1986년 한양대 영어영문(석사). 1992년~2000년 (주)아리스트 선임연구원. 2006년~현재 리버넥스 이사. 관심분야는 소프트웨어 재사용, 소프트웨어 공학, 데이타베이스, 프로그래밍 언어 등

정 안 모

1981년 서울대 물리교육(학사). 1984년~1992년 삼성전자 선임연구원. 1992년~2004년 (주)아리스트 대표이사. 2000년~2004년 이화여대 컴퓨터학과 겸임교수 2004년~현재 리버넥스 대표. 관심분야는 프로그래밍/컴포넌트 모델, 온톨로지, 소프트웨어 공학, 프로그래밍 언어 등