플러그-앤드-플레이 컴포넌트 프레임워크를 위한 조립 계약

이 석 용, 이 경 환
중앙대학교 컴퓨터공학과
argo@object.cau.ac.kr, kwlee@object.cau.ac.kr

Compositional Contract for Plug-and-Play Compoment Framework

SukYong Lee, KyungWhan Lee

요 약

소프트웨어 개발 기술은 컴포넌트의 재사용에 기반한 소프트웨어 개발 방법으로 발전하고 있다. 컴포넌트는 재사용 및 가변성을 촉진시키며, 시스템의 전반적인 복잡도를 감소시켜 준다. 그러나 컴포넌트 조립정보는 디자인시점(design-time)에서 결정되고 특정 컴포넌트 모델과 컴포넌트 실행환경(Component Execution Environment)에 따라 소스코드에 반영되기 때문에, 시스템의 변경 관리 및 동적인 구성등의 소프트웨어 컴포넌트의 장점을 반영하지 못하고 있다. 본 논문에서는 현재 사용되고 있는 컴포넌트 컴포지션의 문제점을 제시하고, 이를 해결하기 위하여 컴포넌트 컴포지션시 필수적 정보들인 컴포넌트간의 상호작용, 컨텍스트 상의 종속성, 메시지 전달 및 호출상의 제약 조건 등의 기본 컴포지션 정보들을 표현하는 컴포지셔널 컨트랙트를 정의한다. 컴포지셔널 컨트랙트는 모든 정보를 컴포넌트 모델 및 환경에 의존적이지 않도록 XML 문서로 정의되므로, 각종 컴포넌트 재사용 및 조립 정보를 여러 환경 및 컴포넌트 모델에 적용시킬수 있다. 또한 이렇게 정의된 컴포지셔널 컨트랙트를 이용하여 컴포넌트 기술의 가장 큰 장점인 플러그-앤드-플레이(Plug-and-Play) 방식의 지원 및 동적 바인딩, 동적 컴포넌트 컴포지션을 지원하는 프레임워크를 제시한다.

1. 서 론

소프트웨어의 품질, 생산성, 효율성을 높이기 위해서 많은 방법론, 프로그래밍 언어들의 변천이 이루어져 왔다. 소프트웨어 개발시의 많은 문제점을 해결하는데에는 "재사용"이라는 개념이 필수적이다. 이러한 재사용의 단위는 현재의 컴포넌트에 이르렀고, 컴포넌트 기반 개발 방법론(CBD)의 핵심은 소프트웨어 시스템을 구성할 때 이미 검증되어 만들어진 소프트웨어 부품, 즉 컴포넌트를 마치 레고 블록을 쌓듯이 조립함으로서 좋은 품질 및 개발비용의 감소를 얻을 수 있다는 것이다. 이러한 플러그-앤드-플레이(Plug and Play) 방식의 소프트웨어 시스템은 또한 손쉽게 컴포넌트를 교체함으로써 얻을 수 있는 유연성, 가용성, 시스템 단위의 재사용성, 유지보수성 등의 이점도 가지고 있다.[1]

이러한 컴포넌트 기반의 재사용을 위해 Java Beans, COM, EJB, CORBA 등의 여러 컴포넌트 모델이 개발되었다. 컴포넌트 모델이란 컴포넌트가 정의되는 방법과 컴포넌트의 조립이 이루어지는 방법을 정의하는 표준이다.[2] 그러나 이러한 컴포넌트 모델들과는 별도로 컴포넌트를 조립하는 방법은 추상적인 컴포넌트의 개념과는 부합하지 않는 공통적인 취약점을 가지고 있다.

본 논문에서는 컴포넌트 컴포지션시 발생하는 문제점들을 도출하고, 이러한 문제점을 해결하기 위한 컴포지셔널 컨트랙트를 제안한다. 그리고, 제안한 컴포지셔널 컨트랙트를 적용 할 수 있는 컴포넌트 프레임워크 구현을 보인다.

2. 관련 기술

2.1 컴포넌트

컴포넌트는 잘 정의된 인터페이스와 은닉된 내부구조를 가지는 소프트웨어 또는 소프트웨어 디자인으로 정의된다.[10] 컴포넌트 기반 시스템은 구축비용의 절감, 품질의 향상등 전반적인 소프트웨어공학의 문제점을 해결할 수 있는 방법으로 각광받고 있으며, 컴포넌트 재사용에 관계된 컴포넌트 기반 개발방법론(CBD) 및 컴포넌트 기반 소프트웨어 엔지니어링 등의 기법이 소개되고 있다.[11][9]

컴포넌트 기반 개발 방법론(Component Based Development)은 요구분석, 설계/디자인, 코딩, 테스트, 적용 등 전체적인 소프트웨어 life-cycle 에 걸쳐 Component의 특성을 적용하는 개발 방법론이다. 특히, 어느 특정 단계만이 아닌 전체적인 소프트웨어 라이프사이클에 적용되어야 컴포넌트가 가지는 여러 이점을 효과적으로 사용할 수 있다[10]. 따라서 전반적인 소프트웨어 구축의 청사진을 제공하는 소프트웨어 아키텍처에 의한 컴포넌트 설계 및 구성은 필수적이다.

2.2 소프트웨어 아키텍처와 컴포넌트

소프트웨어 아키텍처는 시스템의 전체적인 구성을 각각의 컴포넌트로 분해하고, 각 컴포넌트의 기능을 정의하며, 컴포넌트들 간의 상호작용 방식을 규정하는 하이레벨의 시스템 모델링이다. 이러한 소프트웨어 아키텍쳐를 통해 시스템의 이해 및 표준화, 재사용성을 설계단계에서부터 구현, 적용단계까지 일관적으로 관리할 수 있다.[5][6] 컴포넌트 기술은 기존의 추상적 개념이었던 소프트웨어 아키텍쳐 상의 컴포넌트 명세 및 구현에 적용됨으로써 아키텍쳐 기반 소프트웨어 개발에 매우 효과적인 기술로 적용되고 있다. 또한 시스템의 유연성 및 가변성, 타임 투 마켓, 변화 수용성 등의 중요성이 증대됨에 따라, 소프트웨어를 플러그-앤드-플레이 방식으로 구축하기 위한 엔지니어링 기술인 컴포넌트 기반 소프트웨어 엔지니어링(Component Based Software Engineering)이 대두 되었다.[7] 컴포넌트 기술을 소프트웨어 아키텍쳐에 적용하면서 컴포넌트 아키텍쳐 개념이 사용되는데, 이는 컴포넌트를 사용한 소프트웨어 시스템의 디자인과 구축에 관한 일련의 규칙으로 정의할 수 있다. 여기에는 시스템을 구성하는 빌딩블럭(building block)의 타입을 명시하고, 이러한 빌딩블럭이 어떻게 구성되었는가에 대한 정보와 개발 및 유지보수에 대한 조직적 구조가 포함되어야 한다.[8]

2.3 컴포넌트 컴포지션 (Component composition)

컴포넌트 기반 개발 방법론의 기반은 컴포넌트를 소프트웨어의 구축 단위로 사용하는 것이다. 컴포넌트는 항상 위에서 논한 컴포넌트 프레임워크 상에서 특정 의미를 가지기 때문에, 컴포넌트 자체만을 가지고는 플러그-앤-플레이 지원이 가능하지는 않다. 이러한 컴포넌트의 특징에 의해, 컴포넌트 기반 소프트웨어는 다음과 같이 정의 될 수 있다.

Software = Framework + Component + Glue code

여기서 Glue Code 부분이 컴포넌트 컴포지션을 수행하는, 즉 컴포넌트들간의 연결 및 상호작용을 처리하는 부분이다. 그러나 실제 컴포넌트 컴포지션 과정은 컴포넌트 모델 및 해당 모델을 지원하는 컴포넌트 프레임워크에 매우 종속적이다. 또한 컴포넌트들 간의 상호작용은

시스템 구축시 해당 프레임워크를 지원하는 프로그래밍 언어의 소스 코드로 고정이 되므로, 컴포넌트 시스템의 장점을 제한하는 다음과 같은 문제점이 발생한다.

첫째, 컴포넌트들을 실제적으로 조립하는 컴포지션 메커니즘은 프로그래밍 언어에 의해 이루어지고, 따라서 설계정보에서의 각종 컴포지션 정보 명세가 실제 구현에서는 매우 모호하게 나타난다. 따라서 컴포넌트가 특정 시스템에서 어떻게 구성되는가에 대한 정보는 설계 정보에 의존할 수 밖에 없다.

둘째, 소프트웨어 아키텍쳐 및 설계 정보에서는 컴포넌트들과 독립적인 컴포넌트간의 메시지-전달(Message Passing)에 의해 컴포넌트간의 상호작용(Component Collaboration)이 이루어지는데, 실제 시스템 구현에서 사용되는 컴포넌트 간의 메시지 전달은 컴포넌트 인터페이스 호출방법으로 이루어진다. 따라서 설계 정보상의 메시지를 구현상의 메시지로 대응시키는데 어려움이 발생한다.

셋째, 컴포넌트 기반 시스템의 컴포넌트 조립정보는 바이너리 형태로 존재하므로, 이미 구성된 시스템 내부의 컴포넌트 교체를 위해서는 시스템의 소스코드의 수정에 의존 할 수 밖에 없다. 이러한 제약조건은 일부의 개발 툴들에 의해 자동화 되고 있지만, 이러한 자동화 툴들도 각종 리소스 정보를 사용하여 소스코드 자동생성 방법을 사용하므로, 역시 동적 컴포지션, 지연 바인딩(lazy-binding) 등의 컴포넌트 시스템을 구축하는데에는 어려움이 있다.

3. 컴포지셔널 컨트랙트 (Compositional Contract)

3.1. 컴포넌트 컨트랙트 (Component Contract)

컨트랙트(Contract)의 개념은 맨처음 DBC[12]에 의해 제안되었고, DBC는 Eiffle 언어의 메쏘드의 행위를 Pre-condition, Post-Condition, Invarient의 조건에 의한 Assertion 체크로 사용되었다. 이후 UML[13][14], BOCA[15]등에서 컴포넌트 기반 소프트웨어의 아키텍처 및 디자인 정보에 필수적인 요소로 사용되고 있으며, 또한 iContract[16], jContractor[17], JMSAssert[18] 등으로 Java 언어에 적용되고 있다.

컴포넌트 명세는 컴포넌트의 인터페이스 명세와 기능에 관한 설명이다. 그러나 컴포넌트는 시스템에 독립적으로 특정 기능에 대한 잘 정의된 여러 인터페이스를 제공하는 모듈이므로, 실제로 시스템을 구축할 때에는 해당 컴포넌트가 시스템에서 어떻게 구성되고 사용되는지에 대한 정보는 시스템마다 다르게 된다. 이러한 이유로 컴포넌트 시스템에서의 컨트랙트 개념은 기존의 컨트랙트가 객체의 행위(Object behavior)에 초점을 맞춘것에 덧붙여 시스템에서 사용되는 컴포넌트의 역할이 무엇인지, 또 어떻게 다른 컴포넌트와 상호작용이 어떻게 이루어지는지에 대한 명세를 정의하여야 한다[14]. 최근에 Beugnard, Jezequel, Plouzeau, Watkins는 컴포넌트의 컨트랙트를 컴포넌트의 컴포지션 정보가 얼마나 교섭가능한지(negotiable)에 따라 4가지 레벨로 분류하고 각각의 단계를 지원하는 여부에 따라 컴포넌트 기반 소프트웨어의 가변성을 정의하는 인터페이스 명세를 제안하였다[19].

현재의 컴포넌트 컨트랙트 관련 기술의 단점을 살펴보면 다음과 같다.

① UML, DBC에서 사용되는 컨트랙트는 소프트웨어의 디자인 정보를 기술하는데 중점을 두고 있어, 실제 구현시 제대로 반영되는지, 또한 컴포넌트가 컨트랙트에 의해 제대로 동작하는지에 대한 별도의 프로세스가 필요하다.

② iContract, jContractor, JMSAssert 등의 프로그래밍 지원 툴은 behavioral 컨트랙트, 즉 DBC 개념만을 지원하고 있다. 따라서 컴포넌트의 동적인 바인딩, 변경관리 등으로 사용하기에는 부족하다.

③ 컴포넌트간의 동시성 제어, 상호배재, QoS 등을 지원하지 못한다.

④ 별도의 가상머신 또는 소스코드 생성기에 의한 변환을 제공하므로, 빠른 변경 적용이 불가능하다.

본 논문에서 제안하는 컴포넌트 컨트랙트는 앞에서 제시한 컨트랙트가 가져야 할 기능 및 속성에 기반하여 컴포넌트 기반 소프트웨어에서 각 컴포넌트에 필요한 제약사항 및 상호작용 관계를 기술하는 컴포넌트 컨트랙트는 다음과 같이 구성된다.

Compositional Contract=Coordinator+Event Descriptor

① 코디네이터 (Coordinator)

코디네이터는 컴포넌트간의 상호작용을 나타내는 것으로써, 컴포넌트의 삽입, 삭제, 대치, 상호관계 등을 관리할 수 있다.

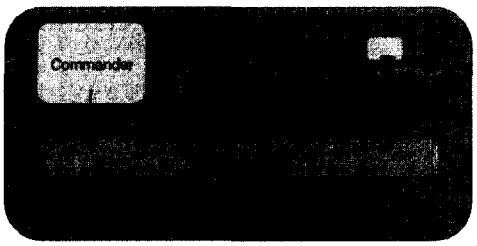
② 이벤트 기술자 (Event descriptor)

이벤트 기술자는 각 컴포넌트가 전달받거나 보낼수 있는 이벤트를 기술하는 것으로써, 실제로 컴포넌트의 인터페이스가 호출될 때 각 매개변수의 제약사항 및 타입, 리턴 값의 제약사항 등을 메타 데이터 형식으로 기술한다. 이 기술자에 의해 컴포넌트간의 상호작용시 의도대로 동작하는지를 사전에 발견함으로써 시스템의 안정성을 높일수 있다.

4. 컴포지셔널 컨트랙트 프레임워크

4.1 프레임워크의 구조

컴포지셔널 컨트랙트 프레임워크는 3장에서 설명한 컴포넌트 컨트랙트를 사용할 수 있도록 하는 프레임워크이다. 그 구조는 다음의 [그림 1]과 같다. 기본적인 구조는 커맨더에 의해 해당 이벤트를 받아들이고, 이 이벤트는 메시지 버스를 사용하여 코디네이터에 의해 해당 컴포넌트에 전달된다. 이때, 시스템에서는 자동적으로 이벤트 기술자의 내용과 전달되는 이벤트를 비교한다. [그림1]의 컴포넌트중 맨마지막 것은, 이 구조로 표현된 부분이 컴포넌트로써 다시 사용될 수 있다는 것을 나타내고 있다.



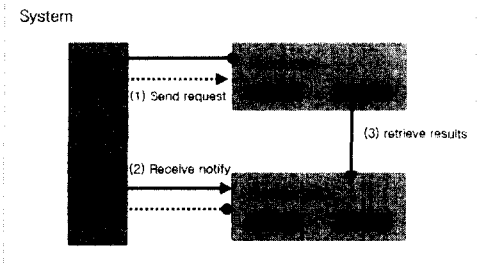
[그림 1] 컴포지셔널 컨트랙트 프레임워크의 구조도

4.1 커맨더 (Commander)

커맨더는 시스템의 usage-driven 디자인을 추상화 한 것으로써, 설계 정보에 표현되는 유즈케이스(Use-Case)의 내용을 직접적으로 현실화 한 것이다. 모든 시스템은 이벤트를 발생시키는 것으로 모델링 할 수 있으므로 이 시스템이 처리할 모든 명령 및 이벤트는 커맨더에 기술될 수 있다.

4.2 이벤트 모델 (Event Model)

전통적인 컴포넌트 시스템은 컴포넌트간의 메시지 교환으로 표현이 된다. 그러나 실제 구현상에서의 컴포넌트 인터페이스 호출은 해당 컴포넌트의 인스턴스 또는 레퍼런스를 가지고 인터페이스를 호출하는 것으로 나타난다. 따라서 또한 이 정보는 소스코드상으로 표현이 되므로, 동적인 컴포지션 등을 지원하기에는 무리가 있다. 따라서 본 프레임워크에서 사용되는 이벤트모델은 이러한 문제점을 해결하기 위하여 인터페이스 호출을 표현하는 이벤트를 정의하고, 인터페이스 호출은 프레임워크가 실행하게 된다. 따라서 시스템의 구축은 컴포넌트와 컴포넌트간의 이벤트를 기술함으로써 가능하여 진다. 이를 지원하기 위하여 각 컴포넌트는 이벤트 생성자(Event Producer)와 이벤트 소모자(Event Consumer)로 구분되어진다. [그림2] 는 이러한 이벤트 모델의 동작을 도식화 한 것이다.



[그림 2] 이벤트 모델의 동작 방식

4.3 코디네이터

코디네이터는 컴포넌트 관리자로써, 컴포넌트의 등록 및 삭제, 인터페이스의 변경, 상호작용 등을 기술하고 있다. 또한 시스템에서 사용되는 각 컴포넌트를 이벤트 생성자 / 소모자 로 등록하는 역할을 한다. 코디네이터는 다음과 같은 컴포넌트 컨트랙트 용법으로 사용된다.

① 행위 컨트랙트 (Behavioral Contract)

코디네이터 상에 표현되는 이벤트 기술자의 내용에 의해 각 인터페이스 호출이 런타임시에 관리된다.

[그림 3] 행위 컨트랙트

② 동시성 제어 (Behavioral Contract)

커맨더에 의해 각 이벤트를 제어하고 이는 프레임워크에 의해 제어되므로 동시성 제어를 할수 있다.

③ 컴포넌트 변경관리

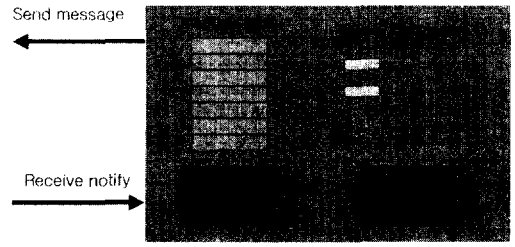
컴포넌트가 변경될시 코디네이터에 의해 제어되고, 또한 인터페이스의 이벤트 기술자에 의해 제어되므로 효과적인 변경관리를 할 수 있다.

④ 타협 (Negotiation)

컴포넌트간의 바인딩은 코디네이터의 내용을 변경함으로써 이루어질 수 있으므로, 동적인 컴포넌트 컴포지션을 지원한다.

4.4 메타 컴포넌트 (Meta-Component)

메타 컴포넌트란 기존의 컴포넌트 모델상의 컴포넌트를 컴포지셔널 컨트랙트 상의 컴포넌트로 사용하기 위한 Wrapper 기능을 하는 컴포넌트 모델이다. 그 구조는 [그림4]에 나타나 있다.



[그림 4] 메타-컴포넌트 구조도

먼저 인터페이스 테이블(Interface table)은 컴포넌트의 인터페이스 기술서로서, 요청 메시지(request message)가 왔을 경우, 해당 요청이 합당한지를 확인하는 정보를 담은 테이블이다.요청 파라미터 버퍼(Request parameter buffer)는 다른 컴포넌트에게 요청을 보낼 때에 해당 파라미터는 컴포넌트 상에서 생성되므로, 데이터 저장소 및 실행속도상의 효율성을 위한 버퍼이다. 응답 파리미터 버퍼(Response parameter buffer)는 해당 요청을 처리한 결과, 컴포넌트의 인터페이스 실행 후 결과 값을 저장하는 버퍼이다. 이렇게 버퍼에 요청 및 응답의 실제 데이터가 들어가며, 컴포넌트 간에는 확인메시지만이 보내어 진다. 이는 불필요한 데이터 저장소 및 실행속도의 효율성을 위해서이다.

5. 결론 및 향후 연구방향

본 논문에서는 현재 이슈가 되고 있는 컴포넌트에 의한 소프트웨어 구축에서, 현재 기술적인 컴포넌트 조립의 문제에 대하여 논하였다. 이를 해결하기 위한 방안으로 컴포넌트간의 상호작용 및 각종 설계정보를 컨트랙트로 표현하고, 컴포넌트의 동적 관리 및 조립을 지원하는 프레임워크를 제시하였다. 이를 통하여, 컴포넌트의 조립을 프로그래밍 언어상의 소스코드가 아닌 XML 문서로 표현되고 동적으로 변경이 가능할 수 잇다있다. 컴포넌트 기반 개발 방법론에서는 시스템에 가장 알맞은 컴포넌트를 선택하는 것도 핵심 프로세스로 규정이 되므로, 본 논문에서 제시한 컴포지셔널 컨트랙트 프레임워크를 개선하여 향후에는 컴포넌트의 조립 및 시스템 테스트를 즉석에서 수행할 수 있는 컴포넌트 시스템 테스트베드(Component system test-bed)를 구축하고자 한다.

참고문헌

[1]Short, K.: Component Based Development and Object Modeling. http:// www.cool.sterling.com/cbd, 1997

[2]Maria Jose Presso. "Declarative Descriptions of Component Models as a Generic Support for Software Composition", France Telecom R&D, 2000.

[3]D'Souza, Desmond and Alan Cameron Wills. Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison-Wesley, 1998.

[4]Lycett, M., & Paul, R.J. "Component-Based Development: Dealing with Operational Aspects of Architecture," 83-92. Proceedings of the 3rd International Workshop on Component-Oriented Programming(WCOP '98). Brussels, Belgium, July 1998.

[5]D. E. Perry and A. L. Wolf. "Foundations for the Study of Software Architectures". ACM SIGSOFT Software Engineering Notes, pages 40-52, October 1992.

[6]Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, Gregory Zelesnik, "Abstractions for Software Architecture and Tools to Support Them". IEEE Transactions on Software Engineering.

[7]rancois Bronsard, Douglas Bryan, W. Kozaczynski, Edy S. Liongosari, Jim Q. Ning, Asgeir Olafsson and John W. Wetterstrand; "Toward software plug-and-play". Proceedings of the 1997 symposium on Symposium on software reusability, Pages 19 - 29, ACM. 1997.

[8]"An Architectural Model : Enabling Component-based Application Development". White paper, Cap Gemini. 1998.

[9]Dabid Sprott, Lawrence Wilkes. "Component Based Development : Application Delivery and Integration Using Componentised Software", Tech. report of Butler Group, 1999.

[10]Peter Herzum, Oliver Sims. "Business Component Factory : a comprehensive overview of component-based development for the enterprise" OMG Press, Wiley Computer Publishing. 1999.

[11]Jean-Guy Schneider and Oscar Nierstrasz, "Components, Scripts and Glue," Software Architectures --- Advances and Applications, Leonor Barroca, Jon Hall and Patrick Hall (Eds.), pp. 13-25, Springer, 1999.

[12] Bertrand Meyer. "Applying 'design by contract'". IEEE Computer, 25(10):40--51, October 1992.

[13]Martin Fowler and Kendall Scott. "UML Distilled: a brief guide to the standard object modeling language. 2nd ed.". Addison Wesley Longman,Inc. October 1999.

[14]Christine Mingins, Yu Liu. "From UML to Design by Contract". Journal of Object Oriented Programming, JOOP. April 2001:6-9, April 2001.

[15]Business Object Component Architecture Revision 1.2. OMG Document 98-07-01, 1998.

[16]Kramer, R. "iContract -the Javadesign by contract tool". Proceedings of the Technology of Object-Oriented Languages and Systems, 1998.

[17]Karaorman, M., U. Hölzle and J.Bruno. "JContractor: A reflective java library to support design by contract". In Proceedings of Meta-Level Architectures and Reflection, volume 1616 of Lecture Notes in Computer Science, July 1999.

[18]Man Machine Systems. "Design by contract for java using JMSassert". http://www.mmsindia.com/DBCForJava.html", 2000.

[19]A. Beugnard, J.-M. Jezequel, and D. Watkins. Making Components Contract Aware. IEEE Computer, 32(7):38--45, July 1999.