자바컴포넌트 상호작용을 위한 환경에 관한 연구

방영환, 정천복, 김혜미, 황선명
대전대학교 컴퓨터공학과
e-mail : bangyh@zeus.taejon.ac.kr
chunbokj@hamail.net
hmkim@jtc.ac.kr
sunhwang@dragon.taejon.ac.kr

Study on the Environment for Java Component Interaction

Young-Hwan Bang, Chun-Bok Jung, Hea-Mi Kim, Sun-Myung Hwang
Dept of Computer Engineering, Taejon University

요 약

소프트웨어를 개발하는 방법으로 기존의 구조적 방법론에서 객체지향 방법론으로 발전하였고 최근 들어 소프트웨어 재사용의 핵심 기술로 인식되고 있는 것이 컴포넌트 소프트웨어기반 개발 방법론이다. 본 논문에서는 컴포넌트 소프트웨어, 컴포넌트 기반개발의 이점, 컴포넌트의 요건과 자바 컴포넌트(자바 클래스 또는 자바 컴포넌트)들 간의 기능, 속성, 연결 등을 동적으로 상호 작용(Interaction)할 수 있는 환경(Environment)을 제안 하려한다. 이와 같은 환경은 이미 만들어진 자바컴포넌트의 컴포넌트를 재사용 및 조립 전에 아키텍쳐의 모델링 도구로서 행위분석(Behavioral analysis)과 인터페이스의 기반설계(Interface Based Design)의 기반을 제공하고 자바 컴포넌트의 인터페이스와 기능을 이해하고 결합형태를 미리 짐작할 수 있는 환경을 제공함으로써 소프트웨어 개발에 소요되는 개발비용을 최소화하고 사용자의 요구를 충족시킬 수 있는 장점을 지니고 있다.

1. 서론

소프트웨어를 개발하는 방법으로 기존의 구조적 방법론에서 객체지향 방법론으로 발전하였고 최근 들어 소프트웨어 재사용의 핵심 기술로 인식되고 있는 것이 컴포넌트 기반 개발 방법론이다. 객체지향 프로그래밍의 문제점을 해결하기 위한 대안으로 소프트웨어 재사용의 핵심기술로 인식되고 있는 컴포넌트 소프트웨어는 일반적으로 객체지향 기술의 원리를 이용해 제작한 소프트웨어 모듈을 의미하는데, 기계부품과 같이 소프트웨어도 하나의 부품으로 제작한 다음 이를 조립해서 보다 복잡한 소프트웨어를 만들 수 있다는 아이디어에서 시작한 개념으로써 컴포넌트기반 개발(CBD : Component Based Development)을 위해 소프트웨어 아키텍처와 명세에 대한 연구, 컴포넌트 프로그래밍, 통합, 검증과 검사를 통한 컴포넌트 표준화 작업등에 관련하여 많은 연구가 되고 있다.[2,4,5] 본 논문에서는 컴포넌트 소프트웨어, 컴포넌트 기반 개발의 이점, 컴포넌트의 요건과 자바 컴포넌트(자바 클래스 또는 자바 컴포넌트)들 간의 기능, 속성, 연결 등을 동적으로 상호 작용(Interaction)할 수 있는 환경(Environment)을 제안하려한다.

2. 관련연구

2.1 컴포넌트 소프트웨어 기반 개발

객체지향 프로그래밍의 문제점을 해결하기 위한 대안으로 소프트웨어 재사용의 핵심 기술로 인식되고 있는 것이 컴포넌트 소프트웨어이다. 소프트웨어 개발에서 컴포넌트는 하드웨어 플랫폼, 운영체제, 소프트웨어가 어떠한 환경에서 구현되는가와 기존 소프트웨어의 한계를 넘어서 개별적으로 동작될 수 있는 조작 단위로 소프트웨어를 개발하는 것이다. 이러한 독립성의 확보는 소프트웨어 개발에 있어 Plug and Play를 실현 할 수 있다. 컴포넌트 소프트웨어 모델이 객체지향 모델의 확장이므로 많은 점에서 유사한 특성을 가진다. 하지만 이러한 유사성에도 불구하고 전통적인 객체는 여러 가지 측면에서 소프트웨어 컴포넌트와 상이하다.[1,3,7,8]

그러한 상이점이 전통적인 객체의 한계를 더욱 명백하게 드러내고 소프트웨어 컴포넌트가 지니고 있는 독특한 특성을 잘 보여준다. 다시 말해서 컴포넌트는 객체모델에 근거를 두고 있다. 이런 의미에서 컴포넌트는 객체라 할 수 있다. 하지만 컴포넌트는 어떤 자발적인 속성을 지닌 상위 객체를 만들 수 있도록 전통적인 객체 모델을 확장한다. 전통적인 객체와는 대조적으로 컴포넌트는 독자적으로 존재할 수 있는데 이 말은 컴파일 과정에 의해 컴포넌트가 개발된 어플리케이션에 종속되지 않는 다는 의미이다. 또한 특정 하드웨어, 소프트웨어, 언어와 독립적으로 운영이 가능하기 때문에 특정 기능을 다른 장소에서 재사용 할 수 있게 해준다. 소프트웨어의 재사용은 소프트웨어 개발자와 객체지향 프로그래밍의 오랜 숙원이었다. 소프트웨어 컴포넌트는 변경 없이 재사용이 가능하도록 설계되었다. 컴포넌트는 외부와 인터페이스만을 통해 동작하므로 외부와 철저히 단절되고 결과 적으로 시스템의 단순성이 높아진다. 또한 객체지향 프로그래밍 개념으로 컴포넌트를 작성해 그 내부에 객체가 존재하더라도 이런 객체는 컴포넌트 내부의 객체만 접근할 수 있고 외부로의 접근은 컴포넌트의 인터페이스를 통해 엄격히 제한된다.[1,7,8]

소프트웨어 개발하는데 있어서 Pre-built 컴포넌트의 조합을 통해서 개발하는 방법이다. 시스템을 유지관리하는 측면에서 표준화된 방법으로 컴포넌트를 추가할 수 있고, 기존의 부분을 대체할 수 있어서 점진적인 진화를 가능하게 한다. 각 컴포넌트의 인터페이스(Interface)를 표준화함으로 얻을 수 있는 이점이다. 그림[1]는 컴포넌트 기반의 소프트웨어 개발의 여러 방법을 보여주고 있다.

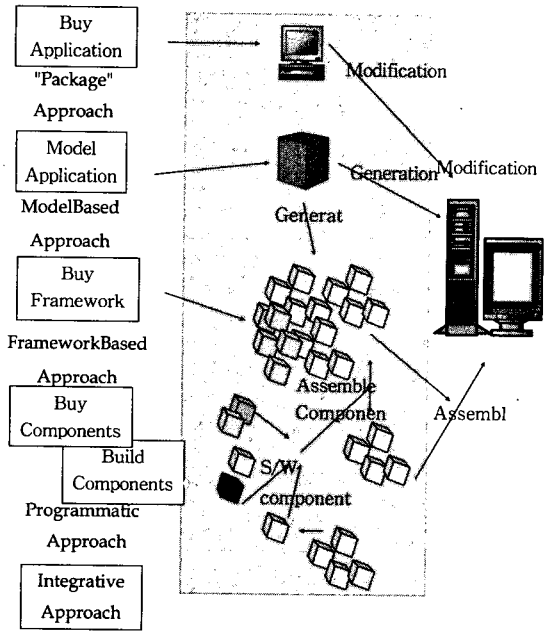

그림1 컴포넌트 개발 접근 방법

컴포넌트는 단순한 관련 클래스들의 조합이 아니라 능동적으로 단위 기능을 수행할 수 있는 소프트웨어로 발전되어 가고 있으며, 모델기반의 컴포넌트 개발로 패러다임이 많이 진화되고 있다. 특히, 컴포넌트를 인터페이스 부분과 구현 부분으로 분리 설계하고 있는 것이 두드러진 특징이다. 컴포넌트가 물리적으로 또한, 논리적이거나 개념적인 형태로 존재하더라도 기본적으로 요구되는 사항과 설계가 개발과정에 포함되어 선택적으로 기대될 수 있는 특징을 가지고 있다[1].

2.2 자바 컴포넌트간의 상호 작용(Interaction)할 수 있는 환경(Environment)의 제안

2.2.1 프로그램 환경

본 논문에서 제안하려는 환경은 컴포넌트를 이용한 어플리케이션 개발에 있었어 컴포넌트의 조립 전에 컴포넌트의 인터페이스와 기능을 미리 확인하며 완벽한 프로그램을 개발할 수 있는 수단을 제공하자는 것이다. 현재 국내에서 사용되어지고 있는 도구들은 코드생성, 설계패턴, 객체지향 방법론, 데이터 모델링, 서버 측의 EJB 콤포넌트를 제공하며 어플리케이션 아키텍처의 모델링 도구로서 행위분석(Behavioral analysis)과 인터페이스 기반설계(Interface Based Design)의 개념등을 사용한다. 이러한 개념들을 이용하여 대부분의 CBD 도구들이 특정 어플리케이션을 구축하기 위해서 바이너리 컴포넌트를 생성하고 생성된 컴포넌트를 조립한다. 컴포넌트의 조립 방법으로는 직접적인 코딩을 통해서 결합하는 형태를 가지고 있으나 구현된 컴포넌트의 인터페이스를 이해하고 관련 컴포넌트들의 상호작용(Interaction)과 정합(Coodination)할 수 있는 지원도구가 아직 까지는 없는 상황이다. 컴포넌트 기술이 보다 강력한 기능을 제공하기 위해서는 컴포넌트의 생성과 조립 및 컴포넌트들의 상호작용(Interaction)과 정합(Coordination)할 수 있는 환경이 제공되어야 한다. 그림[2]는 제안하려는 프로그램 아키텍처 이다.

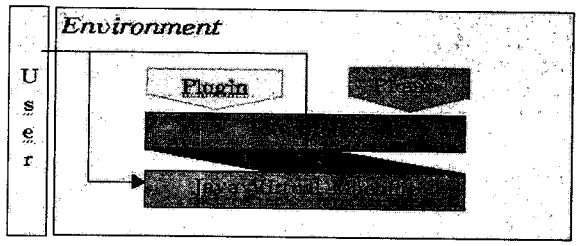

그림 2 프로그램 아키텍처

여기서 플러그인 된 자바 컴포넌트는 일반 자바 클래스 또는 자바 컴포넌트들이고 이들은 인터프리터에 의해서 실시간에 상호 작용(Interaction)과 정합(coordination)이 가능하도록 만들어져야 한다는 것이다. 만들고자 하는 시스템은 여러 자바 컴포넌트와 이것을 여러 각도로 조작 가능한 어떠한 언어(Language)의 선택을 통해서 가능해진다. 이와 같이, 컴포넌트는 시스템의 기능적 분해의 한 단위이며, 이러한 컴포넌트는 서로 만족해야 할 기능(functionality)을 수행하기 위해 서로의 인터페이스를 통해 상호작용(Interaction)을 하게 된다.

2.2.2 프로그램 제한 사항 및 동작원리

컴포넌트 아키텍처 모델인 "C2 Style"을 참고로 하고 "컴포넌트1-포트-커넥터-포트-컴포넌트2"모델을 기본 컴포넌트 모델로 보았다. 메시지(Message), 아키텍처(Archiecture), 포트(Port) 등의 클래스 또는 인터페이스 등의 추가 정의가 필요하며 블랙박스 방식의 소프트웨어 아키텍처의 응용 및 재사용 환경을 위한 소프트웨어 아키텍처 모델 확립 및 구현이 필요하다.

1) 자바 컴포넌트의 기본 아키텍처 모델

가." Component-Port-Connector-Port-Component" 모델

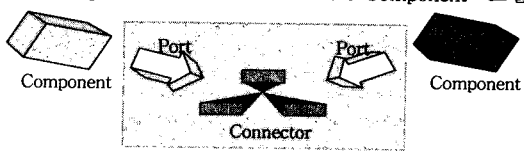

그림3 컴포넌트의 기본 아키텍처 모델

컴포넌트는 포트들로 구성되어 있으며 메시지를 주고받을 때는 반드시 콘넥터(Connector)를 거쳐야 한다. 메시지는 최종적으로 특정 컴포넌트가 받으며 그 메시지에 적당한 포트를 선택하여 그 포트에 추상화되어 있는 서비스를 수행한다.

나. 계층적 콘넥터(Connector)모델

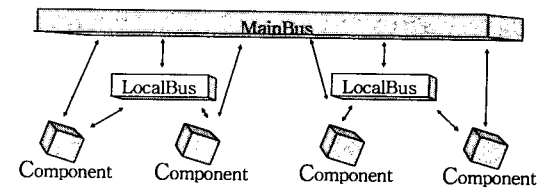

그림 4 계층적 Connector모델

MainBus형 커넥터는 코바의 ORB처럼 모든 컴포넌트들의 중

개자로서 동작하며, 로컬버스 콘넥터(LocalBusConnector)는 비슷한 분류(레이어 또는 서브 시스템 등)의 컴포넌트들의 메시지 중개역할을 수행한다.

2) 동작원리

-계층적 Connector모델 컴포넌트들끼리는 메시지를 주고받음으로서 모든 동작이 이루어진다.

-메시지는 요청(다른 컴포넌트에게 특정 기능을 수행하도록 요청), 응답(그 요청을 수행한 후 반환되는 값), 시그널() 등으로 구분된다.

-컴포넌트들은 직접 메시지를 주고받을 수 없으며 반드시 커넥터에게 메시지를 보내게 된다.

-메시지를 받은 커넥터는 적당한 컴포넌트에게 그 메시지를 보내고, 아니면 다른 커넥터에게 보낸다.

-컴포넌트는 도착한 메시지를 분석한 후에 적절한 행위를 수행한 후 그 결과를 다시 커넥터에게 전송한다. 메시지를 분석한다는 의미는 그 메시지가 요구하는 서비스를 상징하는 포트를 선택하고 그 포트의 서비스를 수행한다는 의미이다. 그 후 결과는 다시 특정 컴포넌트로 전송될 것이다. 이 과정에서 메시지 패싱을 최소화하는 커넥팅 테크닉이 필요하다.

-컴포넌트가 수행하는 서비스는 "포트"로 추상화된다.

-포트는 서비스의 "사전조건(pre condition), 서비스구현(implementation), 사후조건(post condition)" 등이 포함되어 있다.

- 컴포넌트를 만드는 것은 이런 포트들을 생성한 후, 컴포넌트에 포함시키는 과정이다.

- 컴포넌트와 포트의 정보공유를 위해 컴포넌트 컨텍트(Component Context)개념이 필요하다.

2.2.3 인터프리터의 핵심설계요소

인터프리터(Interpreter)는 개발을 위한 분석 및 설계결과, 그리고 구현 시 고려할 점을 제시한다. 먼저 인터프리터의 핵심요소들은 다음과 같다.

- Expression : 인터프리터는 Expression을 받아 그의 Value를 내어주는 함수이다. 따라서, Expression의 효과적인 표현은 효율적인 인터프리터 개발에 있어 필수적이다.
- Environment : environment는 binding들의 Set으로 임의의 expression은 특정 environment에 대해 evaluate 된다.
- Value : expression이 evaluate 된 결과는 value이다.
- Closure : variable에 저장될 수 있고procedure의 argument로 주어질 수 있으며 Procedure의 return value가 될 수도 있다.
- Type : Dynamic type checking을 지원한다.

위의 요소들은 인터프리터의 개발 시 실제 클래스로 표현될 만한 것들이다. 일부는 따로 클래스로 정의하지 않고 다른 클래스의 필드(Field)로 표현될 수도 있지만 이들을 따로 클래스라는 독립적인 단위로 나눈다는 것이 매우 중요한 의미를 갖는다. 자바에서 클래스는 first-class citizen이기 때문에 우리는 어떠한 개념 클래스로 모델링 함으로써 자연스럽게 그 개념을 다룰 수 있는 능력을 얻게 된다. 즉 일종의 추상적인 개념을 구체화하여 이루어질 수 있는 것이다.

여기서 가상의 인터프리터는 자바클래스나 객체를 다룰 수 있는 프로시저를 제공하는 이상 자바가상머신에 대한 조작이 불가피하다. 그래서 기존의 디폴트 클래스 로더(default class loader)에 비해 부가적으로 지원 할 기능에는 Loaded class의 관리, Instance들의 관리, Component들의 관리, Connector들의 관리 등의 구현이 이루어져야한다.

4.결론 및 향후 연구 방향

본 논문에서는 컴포넌트 소프트웨어, 컴포넌트 기반개발의 이점, 컴포넌트의 요건과 자바 컴포넌트(자바 클래스 또는 자바 컴포넌트)들 간의 기능, 속성, 연결 등을 동적으로 상호작용(Interaction)할 수 있는 환경(Environment)을 제안하였다. 이와 같은 환경은 이미 만들어진 자바컴포넌트의 컴포넌트의 재사용 및 조립 전에 아키텍처의 모델링 도구로서 행위분석(Behavioral analysis)과 인터페이스의 기반설계(Interface Based Design)의 기반을 제공하고 자바 컴포넌트의 인터페이스와 기능을 이해하고 결합형태를 미리 짐작할 수 있는 환경을 제공함으로써 소프트웨어 개발에 소요되는 개발비용을 최소화하고 사용자의 요구를 충족시킬 수 있는 장점을 지니고 있다.

소프트웨어의 발전은 현재 컴포넌트라는 개념에 이르게 되었고 관련된 연구가 많이 이루어지고 있다. 다른 소프트웨어들과 마찬가지로 컴포넌트의 품질에 대한 관심도 많이 다루어지고 있다. 향후 연구 방향은 소프트웨어 컴포넌트의 조립 및 컴포넌트의 조립 시 일어날 수 있는 시행착오를 최소화하는 테스팅 기법에 대한 지원도구의 설계를 목표로 한다.

【 참 고 문 헌 】

[1] CBDi Forum, "Component Development Report", Buttler Group, 1999

[2] M. Shaw and D. *Garlan, Software Architecture : Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[3] L. Bass, P. Clenents, and R. Kasman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[4] Desmond F. D'Souza, Alan c. Wills, Objects, Components, and Frameworks with UML, Addison-Wesley, 1998.

[5] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified software Development Process, Addison- wesley, 1998.

[6] Clemens Szyperski, *Component Software-Beyond Object-Oriented Programming*, Addison-Wesley, 1997.

[7] Roger S. Pressman "Software Engineering A Practltiners' Approach" 3rd Ed. McGraw Hill

[8] K.-H.Möller & D,J.Paulish "Software Metrics A practitioner's guide to improved prosuct development", Chapman&Hall Co., New York, 1993.