韓國 컴퓨터情報學會 論文誌
第12卷 第5號, 2007. 11.

2007-12-5-1-13

프레임웍을 위한 효율적인 컴포넌트 모델 설계

임 근*

Effective Component Model Design for Framework

Lim keun*

요 약

프레임웍은 구체적이고 확장 가능한 기반 코드를 가지고 있으며, 설계자가 의도하는 여러 디자인 패턴의 집합으로 구성되어 있다. 클래스의 집합이라는 점에서 클래스 라이브러리와 동일하지만 단순한 클래스들의 모임이라기 보다는 어플리케이션을 개발할 수 있는 아키텍쳐를 제공한다는 점에서 차이가 있다. 프레임워크에서는 컴포넌트 라이브러리를 개발하고 관리하는 것이 매우 중요한 요소이다. 본 논문에서는 프레임웍에 기반한 컴포넌트간 메타관계를 정의함으로서 프레임웍에서 사용될 컴포넌트 모델을 설계하고, 모델을 이용한 적용사례를 제시한다.

Abstract

Framework has constructive and extendable code. It is consist of the variable design collections. In the aspect of the classes's collection, it is similar to the class library. but it is different to provide the architecture which is possible to develop the application rather than the collection of the simple classes. In framework it is very important factor to develop and control the component library. In this paper, we design the usable component model through defining the meta-relation between components and proposed the sample case using this model.

▸ Keyword : CBD(Component based Development), Framework, Component Model.

• 제1저자 : 임 근
• 접수일 : 2007. 10.5, 심사일 : 2007. 10.17, 심사완료일 : 2007. 11.10.
* 을지대학교 의료산업학부 부교수

I. 서 론

객체지향 프로그래밍 기술은 표준 인터페이스와 상속에 의해 코드의 재사용성을 증가시키는 방법을 제시하였고, 이를 상품화한 클래스 라이브러리들은 효과적인 재사용 부품들을 제공해왔다. 따라서 객체지향기술이 재사용성을 증진시킬 것으로 확신해 왔다[1]. 그러나 클래스 라이브러리의 구조상 재사용을 위해서는 라이브러리 구조 이해를 통해 각 인터페이스를 파악한 이후에 클래스 부품을 조립해야한다. 이 과정에서 대형 라이브러리 전체를 이해하고 맴버 클래스 선정하고, 연관성 정의 과정에서 추가부담이 발생된다. 이러한 제한사항의 해결방안으로 객체지향 프레임웍에 대한 연구가 진행되었다[2]. 이 방법도 프레임웍 자체에 대한 이해가 필요하고, 사용방법 등에 관한 지식을 요구한다. 또한 버전관리를 통한 시스템 변화, 환경변화 등의 변경관리가 용이하지 않다. 따라서 본 논문에서는 클래스 라이브러리를 중심으로 재사용하는 방법의 한계와 객체지향 프레임웍이 갖는 제한성을 해결할 수 있는 프레임웍기반에서 컴포넌트 모델 설계 방법과 이를 이용한 적용사례를 나타냈다.

본 논문의 구성은 2장에서 프레임웍 개발 관련 연구 3장에서 컴포넌트 모델설계 및 적용사례 4장에서 비교평가 5장에서 결론 및 향후 연구과제를 기술한다.

II. 관련연구

프레임웍은 추상 클래스 집합과 해당 인스턴스간 상호협력과정의 소프트웨어 시스템 전체 또는 일부분의 재사용 가능한 설계안이라고 할수 있다. 프레임웍은 디자인을 추상클래스로 분리하고, 각종 관련성을 정의함으로서 구축된 아키텍쳐를 결정한다.

2.1 화이트-박스 프레임웍

화이트 박스 프레임웍에서는 상속을 사용한다. 즉 어플리케이션내 클래스를 일반화시킴으로서 화이트 박스 프레임웍을 구축해야 한다. 상속받으려는 수퍼클래스의 재사용코드량을 증가시키면 Template Method나 Factory Method와 같은 패턴을 사용할 수 있다[3].

2.2 블랙 박스 프레임웍

이 방법은 객체 조합에 기반을 두고 있다. 즉 새로운 기능의 추가가 객체의 조합 또는 합성을 통해서 이루어지게 된다. 개발자는 이들 조합된 컴포넌트들을 이용하여 하나의 어플리케이션 행위를 이루게 된다. 객체 조합은 상속보다 더 융통성있는 메카니즘으로 알려져 있는데, 이것은 상속이 컴파일시에 정적으로 이루어지는 반면 조합은 실행시간에 동적으로 이루어지기 때문이다[4].

상속과 달리 컴포넌트들의 객체들에 대한 내부의 자세한 내용을 볼 수 없고, 그와 적합한 타입의 다른 객체에 의해 실행시에 대체될 수 있다. 또한 이는 각 클래스의 캡슐화를 돕고 하나의 작업에 집중하도록 하며 클래스와 클래스 계층도를 작게 유지한다.

2.3 핫 스팟

프레임웍을 기반으로 어플리케이션을 개발하게 되면, 계속해서 반복되어 작성되는 유사코드가 발생한다. 이런 공통의 코드를 공유하기 위해서는 스팟을 정의해야 한다. 이 과정에서 상호 연관이 존재하는데 만일 변경가능한 코드가 어플리케이션에 분산되어 있다면 검색 및 변경이 어렵게 되고, 변경가능 코드가 공통 위치에 있다면 호출시에 해당 코드를 포함하고 있는 객체에서 만들어지기 때문에 프로그램 흐름이 복잡해 진다. 이에 대한 해결방안은 변화되는 코드 부분을 객체내에 캡슐화하는 것이 가능하다[5][6].

2.4 설계패턴

프레임웍은 패턴에 의해 설계되고 문서화될 수 있는데 이것은 각 패턴이 큰 규모의 설계 문제를 작은 부분으로 나누어 해결하는 것이 바람직하다. 설계패턴은 설계 결정사항들을 동기화시킬 수 있으므로 여러개의 마이크로아키텍쳐로 구성된 프레임웍을 문서화하는데 있어 적용될 수 있다[7]. 설계패턴은 목적과 범위에 따라 클래스와 객체 패턴으로 나누어 지며 클래스 패턴은 클래스와 그들의 서브클래스간 관련성을 다루는 패턴들로 상속을 통해 정해진다. 그리고 객체 패턴이란 객체 관련성을 의미하며 실행시 변할 수 있고 좀더 동적인 특성을 갖는다[8].

2.5 컴포넌트 기반 개발

CBD의 목표는 프레임웍상 플러그&플러그가 가능항 소프트웨어 컴포넌트를 가지고 어플리케이션을 구성하는데 있다. 이것은 소프트웨어 아키텍쳐와 소프트웨어 프로세스 모두를 바꿈으로서 재사용을 현실화하였다.

III. 컴포넌트 모델설계 및 관리 방안

프레임웍은 상속과 조합의 두가지 방법에 의해 구성이 되고 확장이 이루어진다. 본 논문에서는 상속을 통한 확장 부분은 다루지 않고, 컴포넌트의 조합을 통한 어플리케이션 생성과 확장에 기반하였다.

3.1 컴포넌트 개발 프로세스

컴포넌트 개발 프로세스는 컴포넌트의 생성, 변경, 확정, 검증과정으로 구분할 수 있다. 생성과정은 실제 컴포넌트의 내용 보다 Identfier의 생성과 관련이 있다. 이것은 컴포넌트의 내용이 여러번 변경, 수정이 이루어진 후 새로운 버전이 등록되고 컴포넌트의 내용이 확정된다[9].

그리고 컴포넌트에 대한 검증을 위해 컴포넌트의 품질이 관련되어 있으므로 이에 대한 형상관리가 정의되어야 한다. 품질 상태는 특정 영역의 컴포넌트 타입에 따라 만들어 놓은 검증 프로세스에 위해서 정의되고, 해당 컴포넌트뿐만 아니라 이와 연관된 다른 컴포넌트에도 기록된다. 계속해서 변경사항이 발생하면 이에 대한 검증은 계속 이루어져야 하다.

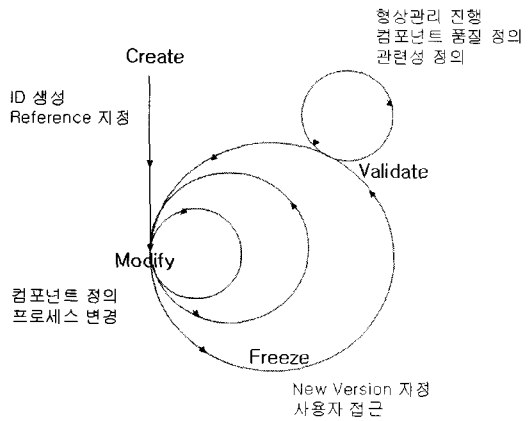

그림 1. 컴포넌트 개발 프로세스
Fig . Component Development Process

3.2 컴포넌트 모델 생성

〈그림 1〉은 컴포넌트를 정의하기 위한 첫 단계로 일반적인 컴포넌트의 클래스를 UML 기호로 보여주고 있다. 이 단계에서는 Identifier를 생성하고, 참조 관련성을 정의한다. 컴포넌트의 개발 과정을 고려하여 기초적인 단계에서 컴포넌트 생성, 확정, 검증 과정에서의 요구사항을 수렴하여 프레임웍에 적용될 컴포넌트의 모델 속성을 정의하였다. 일반적인 컴포넌트의 클래스에서 두가지 중요한 요소를 알 수 있다.

첫째, 컴포넌트들이 서로 의존하고 있는데, 이는 재귀적인 관계를 나타낸다. 즉, 하나의 컴포넌트가 하나 이상의 다른 컴포넌트를 참조하거나, 다른 컴포넌트에 의해서 참조되는 것을 의미한다.

둘째, 컴포넌트의 속성으로 사용자가 입력한 이름 뿐만 아니라 컴포넌트를 구분할 수 있는 유일한 키값을 가지고 있어야 한다. 컴포넌트들의 버전도 계속적인 확장이 이루어진다. 컴포넌트의 의존성이 그 컴포넌트가 참조하고 있는 다른 컴포넌트들의 버전에 따라 달라지고, 컴포넌트의 변경이 이루어져야 한다.

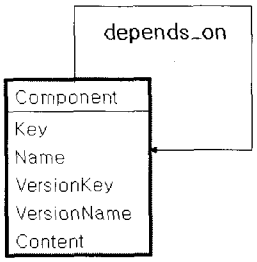

그림 2. 일반적인 컴포넌트 클래스
Fig 2. General Component Class

컴포넌트의 모델로서 〈그림 2〉는 이런 개념들을 만족하지 못한다. 따라서 〈그림 3〉과 같이 Key를 분리시켜 컴포넌트의 Content와 연결되어 있는 독립적인 객체로서 처리하는 것이 바람직하다.

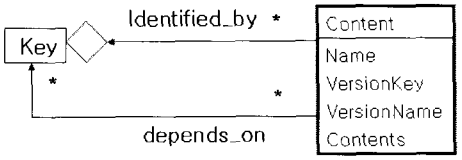

그림 3. 컴포넌트 객체 분리
Fig 3. Component Object Split

컴포넌트의 Content 객체는 하나의 Key에 의해서 구분되는데 이 Key는 해당 컴포넌트 Content 객체의 하나 또는 여러 버전의 Identifier가 된다. 사용자가 입력한 컴포넌트의 이름인 Name 속성은 Content의 여러 버전에 따라 다른 이름을 가질 수 있다. 그러나 Key의 경우는 변하지

않는다. 만약 Key가 변했다면 그것은 새로운 컴포넌트를 생성했다는 의미이다.

VersionKey 속성은 특정 Content의 Idntifier를 나타낸다. 그리고 Key와 Content의 의존성 관계는 다대다의 관계이지만 참조 여부에 따라 구분하기 위해 Content에서 Key로의 단방향성을 갖는다.

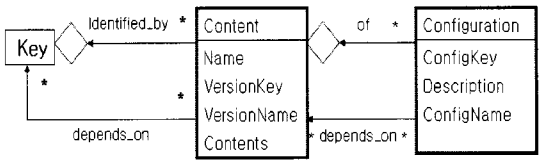

그림 4. Configuration 이 포함된 컴포넌트 모델
Fig 4. Component Model included Configuration

컴포넌트에 대한 품질을 관리하기 위해 형상관리가 필요한데 〈그림 4〉는 컴포넌트의 Content와 Configuration과의 관계를 나타내고 있다. 하나의 컴포넌트에 대한 품질 측정은 해당 Content의 품질 뿐만 아니라 그와 연관된 관계를 나타내고 있다[8].

3.3 컴포넌트의 변경관리

비즈니스 시스템, 클래스 라이브러리 등 대규모의 소프트웨어 컴포넌트들은 연관된 컴포넌트들로 구성되어 있다. 따라서 하나의 컴포넌트는 주로 다른 컴포넌트에 의존하게 되는데 다음과 같은 세가지 타입으로 컴포넌트간 관련성을 분류할 수 있다.

- refers to : 의존성 없이 단순 참조를 할 경우를 의미, 참조되는 내용이 변해도 참조하는 컴포넌트에는 영향을 미치지 않는다
- depends on : 컴포넌트의 품질, 지식등이 다른 컴포넌트에 의존
- consists of : 컴포넌트가 하나 이상의 컴포넌트들로 구성됨

프레임웍은 상속과 조합의 두가지 방법에 의해 구성이 되고 확장이 이루어진다.

본 논문에서는 컴포넌트의 조합을 통한 어플리케이션 생성과 확장에 기반하였다. 〈그림 5〉에서와 같이 프레임웍에서의 컴포넌트는 탬플릿과 핫스팟으로 분류할 수 있다.

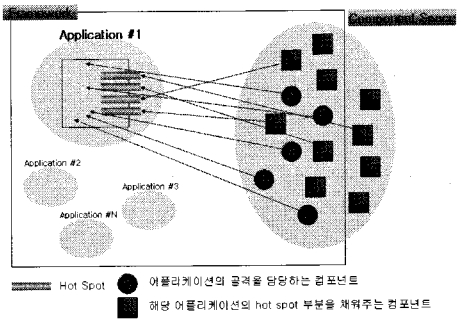

그림 5. 프레임웍 구조
Fig 5. Structure of Framework

어플리케이션의 구조를 이루는 컴포넌트와 핫스팟에 들어가 어플리케이션의 확장 부분을 담당할 컴포넌트에 대한 구분이 필요하게 되며, 또한 여러 핫스팟 중에서 해당 컴포넌트가 들어가게 될 부분이 어디인지에 대한 정보를 가지고 있어야 한다. 〈그림 6〉은 프레임웍에서의 컴포넌트 모델을 나타내고 있다.

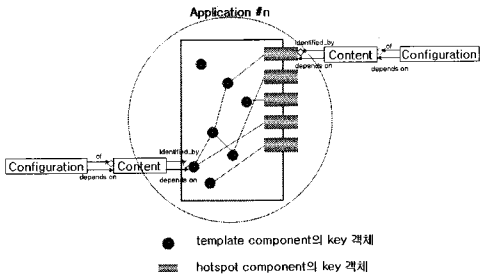

그림 6. 프레임웍에서 컴포넌트 모델
Fig 6. Component Model in Framwork

일반적인 컴포넌트 모델과 비교해 볼 때 핫스팟 컴포넌트인 경우는 Key와 Content, 그리고 Configuration의 관계가 동일하지만, 템플릿 컴포넌트인 경우는 변경 또는 확장이 이루어지지 않고, 어플리케이션 내부에서 변하지 않는 부분이기 때문에 Key는 여러 버전의 Content를 갖지 않고 freeze된 하나의 Content 만을 갖게 된다. 또한 템플릿 컴포넌트도 여러 다른 컴포넌트와의 관계를 맺을 수 있으므로 템플릿 컴포넌트의 Key는 관계를 맺고 있는 다른 Content와의 depends on 의 관계가 성립된다.

앞에서 분류한 컴포넌트간의 관련성 3가지 타입중 템플릿 컴포넌트간에는 refers to가 템플릿 컴포넌트와 핫스팟 컴포넌트 사이에는 depends on 관계가 된다.

형상관리는 각 영역마다 정해놓은 품질 기준과 프로세스에 따라 다르게 이루어지게 되는데, 이런 변경이 일어날 때

마다 해당 컴포넌트의 형상관리 뿐만 아니라 이와 관련된 다른 컴포넌트의 형상관리도 수행된다.

3.4 컴포넌트 검증

컴포넌트는 해당 분야나 프로젝트등에 따라 여러 종류의 타입이 존재한다. 이런 여러 종류의 컴포넌트에 대한 검증을 위해 일반적인 품질 표준 기준을 적용한다는 것은 불가능하다. 따라서 이 문제를 해결하기 위해 객체지향 개념에서 사용되는 일반화-상세화(Gen/Spec)의 개념을 적용한다. 상위 레벨의 컴포넌트의 검증을 위해서는 보다 일반적인 기준을 정한다. 그리고 하위 레벨의 컴포넌트의 검증을 위해서는 보다 일반적인 기준을 정한다. 그리고 하위 레벨의 경우는 상위 레벨에서 적용했던 기준을 만족하는 보다 상세한 기준을 정의하고 그것에 따른 검증을 하게 된다. 결국 컴포넌트의 개발 프로세스의 목표는 요구되는 품질 상태를 만족하는 컴포넌트를 만들어 내는 것이다. 컴포넌트는 미리 정의된 컴포넌트의 타입 중의 하나여야 하며, 각 컴포넌트 타입에 대한 규격이 정의되어 있어야 한다[10][11].

3.5 프로토타입 시스템

프레임웍에서의 분석은 주어진 영역내의 모든 어플리케이션들에 대한 공통적인 요구와 그에 따른 객체들을 식별하는 것이다. 이는 3개의 어플리케이션에 대한 분석을 통하여 얻어질 수 있는데, 어플리케이션의 수가 이보다 많으면 복잡도나 효율적인 면에서 좋지 않고, 적으면 일반적인 공통사항을 얻기가 어렵다. 프레임웍의 분석과정도 기존의 객체지향에서의 분석과정과 비슷해서 공통의 요구사항은 기능정의와 Use Case 모델을 통해 표현을 하고 공통의 객체들은 정적인 객체를 추출하고 그들간의 관계를 정의하는 정적 객체모델과 객체들간의 상호 협력의 관계를 기능중심으로 표현한 Collaboration 다이어그램으로 나타낸다. 그리고 이때, 공통부분에 대해서는 추상화 레벨을 높임으로써 좀더 유연한 소프트웨어 아키텍처를 제공할 수 있다. 〈그림 7〉과 같이 프레임웍은 컴포넌트들과 이들의 메타관계들로 이루어지게 된다. 영역분석 과정에서 나온 요구사항과 분석사항으로부터 실세계 객체들을 시스템에서의 객체들로 표현하여 개선된 객체를 추출하고 객체에 대한 책임을 부여하면 객체들간의 연결관계를 정의함으로써 각 객체에 대한 공통구조를 추출한다.

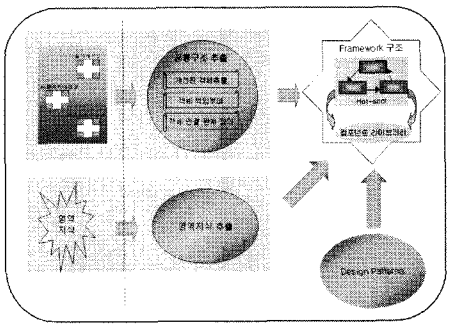

그림 7. 설계지원 단계
Fig. Phase of Design support

그리고 여기에 영역 전문가로부터 나온 영역 지식과, 이미 증명된 해결안을 제시해 주는 디자인 패턴을 적절히 사용하여 프레임웍의 구조를 이루게 된다. 프레임웍에서 영역내의 공통지식을 추출하는 것은 추상화의 과정을 거치는데 여기에서 표현되는 객체는 하나의 컴포넌트로 간주할 수 있다.

- 적용 사례

〈그림 8〉은 본 논문의 적용 예로 제시하는 BBS 시스템 모델링이다. 정적모델에서는 특히 Action의 행위가 선택하는 기능에 따라 다양하게 변환된다. 이것은 설계 패턴중 객체의 행위가 상태에 따라 변화될 수 있기 때문이다.

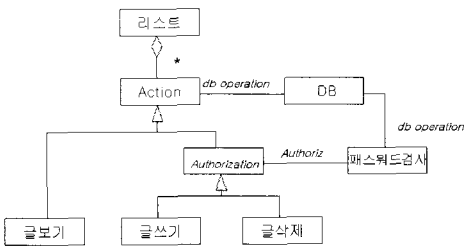

그림 8. BBS의 정적 객체모델
Fig 8. Static object model of BBS

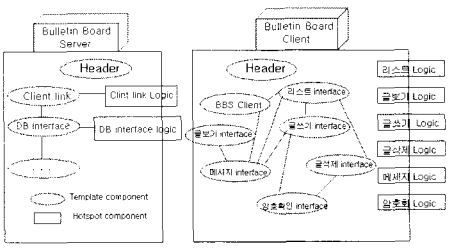

그림 9. BBS의 key 객체
Fig 9. Key object of BBS

분석과정에서의 결과를 기반으로 컴포넌트를 추출하고 이에 대한 관련성을 정의함으로서 참조할 수 있는 단계의 key를 생성한다. 〈그림 9〉는 BBS 시스템의 서버와 클라이언트에서 사용되는 Key 객체와 객체간 관계를 보이고 있다.

〈그림 10〉은 서버와 클라이언트 각각의 Collaboration 다이어그램이다.

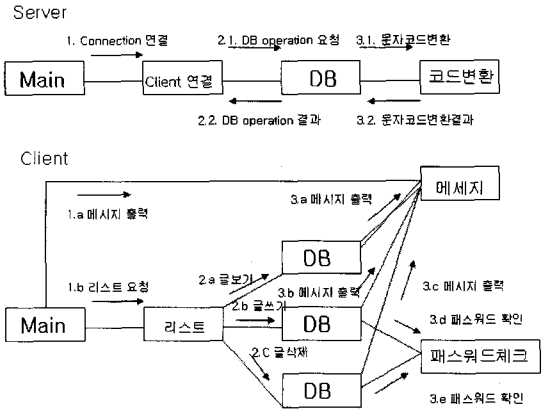

그림 10. BBS에서 서버와 클라이언트 Collaboration 다이어그램
Fig 10. Server and Client of BBS

IV. 비교평가

객체지향 기술은 바이너리 형태의 파일을 연결하는 표준이 없으며, 각 객체는 동일 한 컴파일러를

사용해야 하는 제한성이 있다. 따라서 상속받은 상위 객체의 소스코드까지 접근해야하는 부담을 갖는다. 또한 이기종 언어 간에 객체를 사용하는 것은 불가능하기 때문에 호환성의 한계를 내포하고 있다. 이에 비하여 컴포넌트는 외부와 인터페이스만을 통해 동작하므로 외부와 단절되고 결과적으로 시스템의 단순성을 유지할 수 있다. 시스템의 확장성에 있어서는 클래스 라이브러리의 경우 상속을 통한 제한적 확장만 지원하고, 객체지향 프레임웍에서는 핫 스팟의 중첩을 통한 제한적 확장만 지원하는 반면 본 논문에서는 컴포넌트 합성에 의한 다양한 확장성을 지원한다. 재사용 단위의 분류성에 있어서도 기존 방법은 단순 분류를 통한 제한적 기능만을 지원하는 반면 본 논문에서는 객체지향 프레임웍및, 템플릿, 핫 스팜 컴포넌트 등 다양한 분류를 통해서 가장 적합한 재사용 단위를 지원할 수 있는 장점을 갖는다. 〈표 1〉은 사용성, 확장성, 버전관리 등으로 고려한 각 사례들의 기능비교이다.

표 1. 각 사례의 가능성 비교
Table 1. Compare of each examples efficiency

	클래스 라이브러리	객체지향 프레임웍	본 논문
사용성	계층적 클래스 이해	해당 영역의 부분만 이해	해당 영역의 부분 이해 기존 컴포넌트의 합성가능
확장성	상속을통한 제한적 확장	hot spot overriding을 통한 제한적 확장	컴포넌트 합성에 의한 다양한 확장지원 가능
이해성	코드 수준의 이해가 난해	객체 프레임웍의 이해 어려움	인터페이스를 통한 용이한 이해 지원가능
분류성	클래스 계층도를 통한 단순 분류만 가능	포괄적 분류로 인한 상세단계 분류가 필요함	객체지향 프레임웍 template 컴포넌트 hot spot 컴포넌트등 다양한 분류 가능
유지 보수성	모델적용의 융통성 부족	제공된 패턴 이외 유지보수 낮음	검증을 통한 부품을 사용하므로 효과적 적용 가능
영역 적용성	제한적 적용만 가능	기본 패턴 적용만 가능	재사용단위 확장이 가능하므로 다양한 적용
버전 관리	없음	제한적 적용	컴포넌트에 대한 버전관리 가능

V. 결론 및 향후 연구과제

컴포넌트 라이브러리를 객체지향 기술 기반이 아닌 컴포넌트를 기반으로 개발함으로서 새로운 컴포넌트를 독립적으로 개발하거나 또는 이미 검증된 컴포넌트를 사용하여 구축시간과 코딩에러, 유지보수 비용 측면에서 많은 이점을 가지게 되었다. 객체지향 기술에서 내포하고 있는 문제점으로 바이너리 형태의 파일을 연결하는 표준이 없으며 각 객체는 동일한 컴파일러를 사용해야 한다. 그 결과 사용할 수 있는 객체 라이브러리는 대부분 소스 코드까지 포함하게 된다. 다음으로 이종 언어간 객체 사용이 불가능하다는 것도 문제점이라고 할 수 있다. 이에 비하여 컴포넌트는 외부와 인터페이스만을 통해 동작하므로 외부와 독립성이 유지되어 결과적으로 시스템의 단순성이 높아진다. 또한 컴포넌트는 인터페이스와도 독립성이 유지되므로 내부 구현이 변경되어도 기존 인터페이스를 사용하던 클라이언트는 그대로 상속할수 있게 된다. 따라서 이러한 컴포넌트관리를 통해 컴포넌트 라이브러리를 개발할 경우 사용성과 확장성 및 이해성을 극대화할 수 있고, 관리적인 측면에서 버전관리 및 컴포넌트 분류의 용이성을 지원할 수 있으며 보다 유연하고 안정적인 프레임웍을 구축할 수 있을 것으로 기대한다.

향후 연구과제로 자바빈즈와 같은 바이너리 수준의 컴포넌트의 개발을 통한 연구가 필요하며, 컴포넌트 라이브러리 도구에 대한 전반적인 프레임웍 구축을 위해 지원도구간 연계성 연구가 필요하다.

참고문헌

[1] 임근, 권영만 , 객체모델에 대한 형식명세로의변환 방법, 한국컴퓨터정보학회논문지 제8권 4호 2003년 12월

[2] Rumbaugh J., Blaha, M., Premeralni, W., Eddy, F. and Lorensen, W. "Objected Modeling and Design." Prentice Hall, 1991.

[3] Desmond Francis D'Souza, Alan Cameron Wills. Objects components and frameworks with UML, Addison Wesley Longman, Inc., 1999

[4] Wolfgang pree, "Design patterns for object oriented software development", addison-wesley, 1995.

[5] Frank Buschmann and Regine Meunier, " A System of Patterns", pattern language of programing language 2, 1995.

[6] M, Thomas.C.Carson and J.M Hellerstein. "Creating a Customized Access Method for Bloworld". Proc. 16th International Conference on Data Engineering. pp.82-92. 2000

[7] Dauglas C. Schmidt, "Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software", Communication of ACM, October 1995.

[8] 윤희환, CBD 환경에서 컴포넌트의 재사용성 측정 메트릭스, 한국컴퓨터정보학회논문지 제10권 4호 2005년 9월.

[9] 정규장, 웹 기반 공간데이터 공통 컴포넌트 설계 기법, 한국컴퓨터정보학회논문지 제9권 1호 2004년 3월.

[10] Jiri Soukup, "Implementing pattern language of program design," Addison-wesley, 1994.

[11] C. Szyperski, Component software, Addison-Wesley, 1998

저 자 소 개

임 근

1998년 중앙대학교 컴퓨터공학과 공학박사

1992년 - 현재 을지대학교 의료산업학부 부교수

관심분야 : 정보검색, 데이터마이닝, 재사용 방법